新建文件夹后在文件内打开命令行，输入：

Cnpm init -y

Cnpm i vue --save

Src 放置源码

Dist 打包之后的文件放置的地方

Vue为一个构造函数使用时需要使用new

虚拟dom：内存中有一个和实际dom树结构对应的对象

比对算法：为diff算法（同级比较）

模板（template）

<Div id =”box”> {{msg}} </div>

New Vue ({ el: “ #box ”,data: {msg: “hello world” }})

{{差值表达式}}

其实Vue内部做了dom操作

1、el:挂载：

el: 挂载的元素

new Vue ({data: {msg: “hello world” }}).$mount(“#box”)

2、data:放置数据

3、template：放置模板（优先级高于挂载元素的模板）

数据驱动：数据变化时视图会自动变化

基于MVVM的框架

M:model 模型

V:view 视图

VM: vue的实例，实现数据变化，视图自动变化、

v-model 双向绑定命令

数据变化，视图会变，视图变化，数据也会变

修饰符：

v-model.number 将输入内容约束为数值型

v-model.lazy 失去焦点时才会变化

v-model.trim 删除字符串前后空格

v-bind 单向绑定，数据变了，视图会变，视图变了，数据不会变

v-bind:属性=”变量”

v-bind:title=”color”可以简写成:title=”color”;

v-bind:style=”{background:color}” 需要写成对象格式

v-if 为眞值，渲染，为假值，不渲染

v-else 与v-if同时使用

v-show 为真值，显示，为假值，隐藏

Show是会自动改变元素的display属性来实现显示和隐藏

v-if和v-show的区别：

1、v-if为惰性的，只有值为真时才会进行渲染，而v-show则会直接渲染，通过改变其 display属性来实现显示和隐藏

2、v-if会确保在切换过程中条件块中的事件监听器和子组件适当的销毁和重建

v-for

数组 （元素，下标） in arr

对象 （值，键，下标）in obj

数字 x in n

v-for=”n in 30” 从1到30的所有数

在进行嵌套时，要保证结构正确

v-on 事件绑定，简称@

事件处理的函数和自定义的方法都需要放在methods中(el,data,template,methods)

v-on:click.stop 阻止冒泡

v-on:click.prevent 阻止默认行为

v-on:keyup.enter

v-on:keyup.esc

v-cloak 防止绑定内容为执行时显示的花括号闪烁

v-once 只能绑定一次

v-html 渲染html标记

v-pre 不解析差值表达式

v-text 渲染文本，更新元素的textContent

Includes ES6中字符串的查找方法（string.includes(“aaaaaa”)）

Filters:{my(){}} 过滤器，用来限定格式

过滤器名称（value）{ return 返回值}

Html中用{{money | my}} money为值，my为过滤器

Mixins:[obj1,obj2] 可以把多个组件公用的数据或方法抽离出来，实现复用

Directives 自定义指令

directives：{one(el,binding){}} el为元素，binding为绑定信息

Html中使用v-one来标记

Computed 计算属性

可以把函数的返回值当做计算值去使用

语法：

属性名称(){

return 函数返回的值就是计算属性的值

}

属性：{

Get() {

Return 值就是计算属性的值

},

Set(v) {

v就可以给计算属性赋值

}

}

动态绑定类

对象类型 v-bind:class=”{‘类名1’: boolean,‘类名2’:布尔值表达式}”

如果键值（类名）是个变量的话需要使用[]

数组类型 v-bind:class=”[‘类名1’,’类名2’......]“

组件：component 页面上复用的一部分

全局组件

Vue.component(“组件名称”,{template:’<div>asdfsdfsda</div>’})

组件名称定义为驼峰式时，div中的标签用”-”连接

组件中的data：

data(){ //每个组件的实例返回一个对象，这样可以保证每个实例数据的独立

return {}

}

局部注册的组件

var 组件的名称 = {

template:’<div>asdfsdfsda</div>’

}

局部注册的组件一定要在组件中注册

new Vue({components:{oneComponent:oneComponent}})

组件上的事件是无法响应的，需要加.native修饰符

父组件给子组件传值：

加v-bind可以识别值的类型，但是不加v-bind时默认为字符串

<组件名称 属性=”传给组件的值”></组件名称>

传的值是字符串

<组件的名称 v-bind:属性=”传给子组件的值(各种类型)”></>

子组件中使用props:[“属性”]来获取属性值，在子组件中可以直接使用{{属性名}}

当传来的值是对象时，需要用props:{ a:Number } 属性名:类型

props:{

a:{ //属性名称

type : Number, //类型

default : 20, //默认值

required: true, //必须传参

validator:function(){} //自定义验证函数，验证失败时控制台会报错

}

}

值可以是变量，在父组件data需定义该变量

created 在实例创建完成后会被立即调用

子组件向父组件传值

this.$emit(“事件名称”,所传数据)

this.$forceUpdate 强制刷新

1. 发送数据this.$emit(“事件名称”,”所传数据”)
2. 标签中写监听事件
3. 父组件methods写监听函数

非父子组件传值两种方法：

1. 通过共同的父组件进行传值
2. 通过vuebus传值（一个空的vue实例） $on $emit

v-for要指定一个key值，并且这个值不能重复，最好不要变化（对每个列表内容设个id）

<component :is=”componentName”></component>

在data中加componentName:组件名称

Keep-alive 可以进行缓存组件里的内容

通过插槽分发内容

Slot(插槽)

匿名插槽（没有名字的插槽）

具名插槽（有名字的插槽）

作用域插槽（可以传递数据）

匿名用法：

<组件>放置插槽的内容</组件>

组件中加<slot></slot>

具名用法：

<组件><标签 slot=”aaa”></标签></组件>

组件中<slot name=”aaa”></slot>

作用域用法：

<组件><标签 slot=”aaa” slot-scope=”props”>{{props.m}}</标签></组件>

组件中

<slot name=”xxx” v-bind:m=”n”></slot>

Data(){return {n:”abc”}}

当前的写法：v-slot:插槽名称=”props” {{props.变量}}

v-slot必须写在template标签，html写在template标签中，v-slot:可以简写成#

生命周期的钩子函数

定义：就是在组件的整个生命周期里，自动调用的函数

BeforeUpdate和Updated监控组件内所有数据的变化

BeforeUpdate 数据更新之前调动

Updated 数据更新之后调用

Watch监听特定值的变化 //只能监控简单值的变化，不能监控对象的变化

watch:{

监控的值(newV，oldV){

}

}

对象必须深度监控

watch:{

obj:{

Handler(新值，旧值){

}，

deep:true, //深度监控

Immediate:true //立即执行

}

}

服务器端渲染（首屏到达时间更快，更好的seo优化）

客户端渲染（引入外部js文件渲染）

1. beforeCreate

（1）在数据调用之前，拿不到data中的数据

（2）在实例初始化之后，在数据观测和event/watcher 事件配置之前被调用

2、created

（1）在实例创建之后立即调用，挂载阶段之前

（2）已完成数据观测

（3）可访问数据，无法访问节点

（4）数据变化之后视图也会刷新，但是beforeUpdate和updated不会被调用

（5）在节点卸载之前只会执行一次

3、beforeMount

（1）在挂载节点之前调用，只会执行一次

（2）服务器端渲染（ssr）不会被执行

（3）编辑模板已经完毕，虚拟dom以及存在，可以访问数据，但改变护具不会触 发beforeUpdate和updated

（4）没有实际节点，无法访问到节点

4、mounted

（1）挂载真正的节点，才有真实的dom，改变数据会触发beforeUpdate和update

5、beforeUpdate

6、updated

（1）在更新数据时有死循环的可能

7、beforeDestroy //清理资源防止内存的泄漏

this.$destroy() 可以销毁组件

（1）组件销毁之前调用

8、destroyed

（1）组件销毁之后调用

（2）Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。

（3）需要手动清除计时器

9、activated

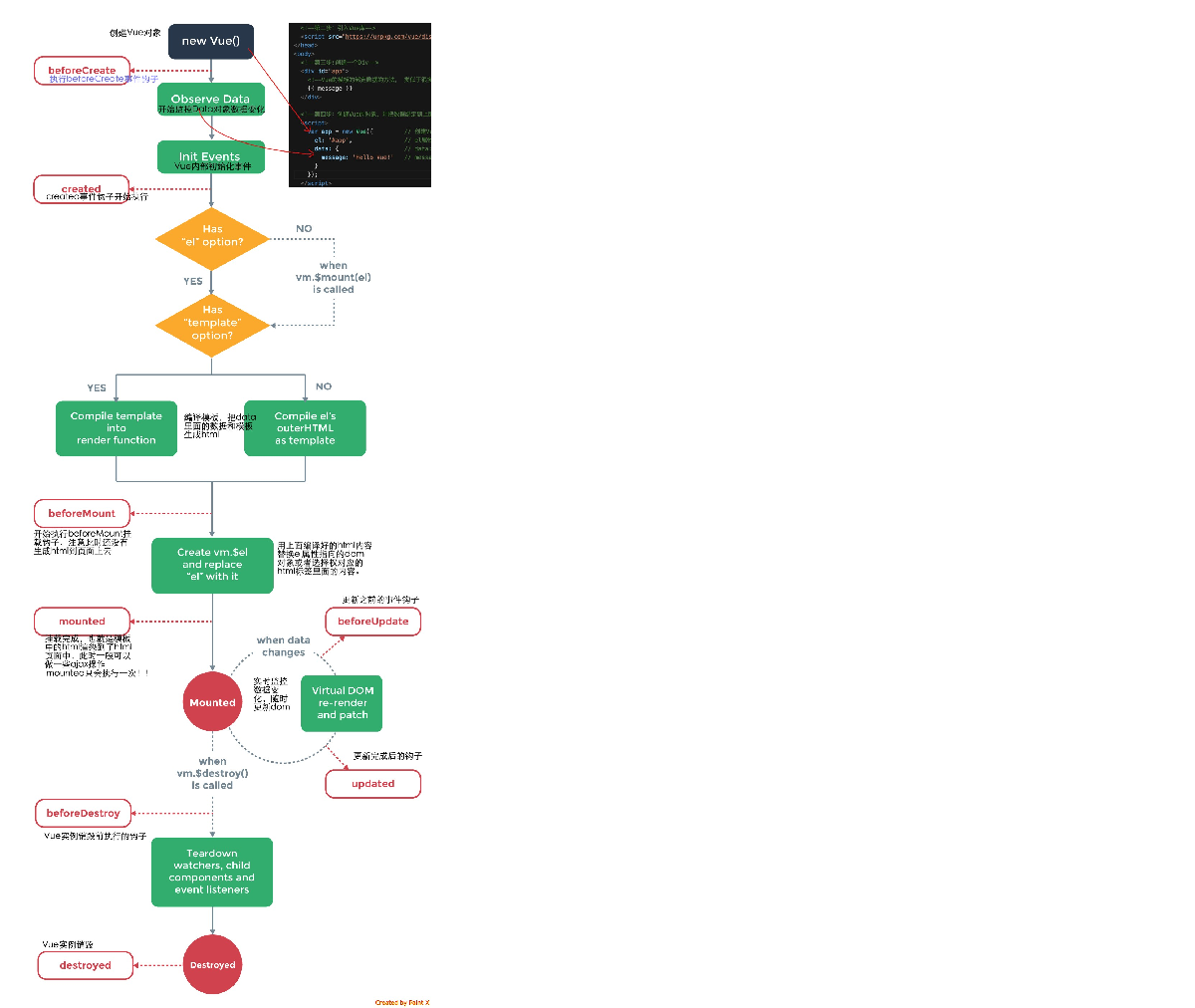
（1）keep-alive组件激活时调用

10、deactivated

（1）keep-alive组件停用时调用

11、render

（1）Render(createElement) 会覆盖系统自动调用的render，一般不手动调用



Vue/cli 脚手架环境（webpack）

1. 支持热更新
2. 单文件组件（.vue） vue-loader
3. 支持es6的导入导出
   1. Import 标识符 from “路径” 导入
   2. Export default 标识符 导出

安装：cnpm i @vue/cli -g

验证版本： vue -V

建立项目：vue create 文件名

运行项目：npm run server

更改配置：在文件夹下新建文件Vue.config.js

module.exports = {

devServer: {

host:'0.0.0.0'

}

}

配置之后重新运行，手机与电脑同一局域网，都可访问

每一个组件文件都是.vue结尾的文件，分成三部分

template、script、style

脚手架的目录结构

Src 源码 assets下的

SPA 单页应用

@表示src目录

npm i axios --save

Import axios from ‘axios’

Axios.get(url,params:{参数}).then().catch() axios.get方法，返回之时promise

Axios.post(url,{参数}).then().catch() axios.post方法，返回值是promise

正向代理：（Vue.config.js）

Module.exports = {

devServer : {

Proxy:{

“/hd”: {

“target” : ”http://localhost:3000”,

“changeOrigin” : true,

“pathRewrite”: {

}

}

}

}

}

Json-server json文件 --prot 端口号

只引入一次axios的方式

入口文件 main.js

Import axios from ‘axios’

Vue.prototype.axios = axios;

组件中使用this.axios

Json-server查找

精确查找

Url？字段(field) = 值

模糊查找

Url? 字段\_like=值

全文查找

Url？q=值

路由：前端路由，根据hash值来切换不同的组件

Hashchange 当hash变化时触发

Vue.use() 注册插件,调用了插件中的install方法进行全局注册

redirect 重定向

Alias 别名

1. 引入 import VueRouter from ‘vue-router’
2. 注册插件 Vue.use(VueRouter)
3. 实例化

New VueRouter({ //这个实例叫router对象

routes:[ { path:’xxx’,component:’yyy’ }.... ] //数组中的每一项叫route对象

//path 路径

//component 组件

//redirect 重定向

//alias 别名

})

1. 在根组件里进行注册 new Vue(router:router)
2. 在组件中写入<router-link to=”/one” tag=”渲染到页面的标签,默认为a”></router-link> <router-view />

具体操作:

1. 先在components文件夹下写组件
2. 在router.js中导入，

import VueRouter from 'vue-router'

import Vue from 'vue'

import one from './components/one'

import two from './components/two'

import three from './components/three'

Vue.use(VueRouter)

1. 在router.js中写

New VueRouter({ //这个实例叫router对象

mode:’history’, //选择模式，history为历史模式，url中没有#

routes:[ { path:’xxx’,component:’yyy’ }.... ] //数组中的每一项叫route对象

//path 路径

//component 组件

//redirect 重定向

//alias 别名

//name 路由名称

})

1. Main.js中到入并注册
2. 在app.vue中加入<router-view/>标签，通过<router-link to=”/one” tag=”渲染到 页面的标签,默认为a”></router-link> 切换页面

路由有两种模式：hash模式和history模式

异步组件 component: ( )=>import(‘./views/about.vue’)

router-link-exact-active 当前激活的

404页面 使用path:”\*\*”,并且把该项放在routes的最后

cnpm i node-sass sass-loader --save

图标库的使用：

1. 将图标库的font-awesome包里的css,fonts文件夹到public目录下
2. 在public/index.html <link rel=”stylesheet” href=”css/font-awesome.css”>

style标签上要写个scoped 样式仅仅是在当前组件里生效，不会影响别的组件

引入css样式的方法

1. 在<style lang=”scss”>中写@import “url”; //写scss效率更高
2. 在script中写import “url”; export default{}

移动端适配：

在index页面头部写document.documentElement.style.font-size=document.documentElement.clientWidth/15+”px”;

//根据选择的机型来求，使fontsize为整数，如750的除以15,640的除以16

在style中写$sc=25; //25为其上求出来的fon-tsize

在scss中使用可以用100/$sc+rem;

Babel-loader .vue 进行编译为vue可识别的组件形式

子路由：

在router.vue 中的routes中的某一项下写children:[{path:”/上一级路径/自己的路径”,component:”组件”}]

在子路由的组件中写<router-linker to=”/one/qd”></router-linker>

路由传参

/path:参数

/path/aa

如何取得路由的参数：

this.$route去的当前的path匹配的路由对象

取得/:之后的参数 this.$route.params.参数 取得路由的参数

取得？之后的参数 this.$route.query.参数

当一个组件路由参数变化的时候，组件并没有卸载

路由传参2：

在路由表中props：true

在组件中：props:[]

编程式导航

this.$route.go(-1)

this.$route.push()

this.$router.push({name:’route的name属性的值’,params:{传的参数}})

localeCompare() 比较两个字符串的大小，按字母排序

meta 路由的元数据

导航守卫

全局守卫

beforeEach((to,from,next) => {})

afterEach((to,from) => {})

路由独享的守卫

beforeEnter((to,from,next) => {})

组件内的守卫

beforeRouteEnter 执行时间别beforeCreate早，但是通过next的回调函数可以拿 到实例，就可以访问数据了

beforeRouteUpdate

beforeRouteLeave

登录和退出操作

1. 把路由加上meta:{requiresAuth:true}
2. 给route.js 加
3. 制作一个login组件，在路由表中设置好
   1. 登录组件用来判断登录是否成功
   2. 成功往sessionStroage.setItem(“users”,用户名)
   3. 把用户名发给父组件
   4. 跳转到路由登陆前的组件
4. app.vue
   1. router-view @child=”handler”接收用户名，显示退出按钮 显示当前登录的用户名
   2. 点击退出时，清空sessionStorage
   3. 清空登录的用户名
   4. 调到首页

命名视图<router-view name=”routerview 的名字”>

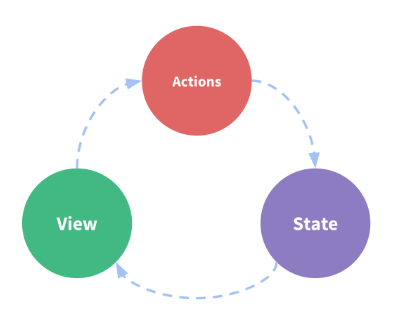
new Data(value).toLocaleString() 将时间戳变成本地实现

实例方法：$on $emit $destroy $forceUpdate $mount

实例属性： $refs

Vuex是一个状态管理工具 核心是store（仓库）

仓库是容器，存放大部分的状态（唯一的仓库）



存数据：

1. import Vue from ‘vue’

Import Vuex from ‘vuex’

1. 使用 Vue.use(Vuex)
2. 实例化

New Vuex.Store({ state:{ 数据}, mutations:{change(state,payload){ state.a = payload;}} })

//state存放数据，mutations存放同步方法

1. 在main.js中注册 new Vue({store:store})

取数据 this.$store.state.变量

改数据

同步的方法放到store的mutations里

Mutations:{方法名(state,参数){state.变量 = 参数}}

调用： this.$store.commit(“mutations里的方法名”,修改后的数据)

export default 导出

Import 名字随便起 from ‘文件路径’ 导入

Export const NUM =5;

Import {NUM} from ‘文件路径’

辅助函数：mapState 写在computed里面 ...mapState([“a”])

mapMutations

mapGetters

类似计算属性：getters

Getters:{属性(state){return state.属性? “” : “”}}

使用:this.$store.getters.属性

异步的数据

方法名(context,payload){

异步的callback里面

context.commit(“mutations里面的同步的方法名”，payload)

}

调用方法 this.$store.dispath(“异步里的方法名”,参数)

vue-jsonp 适合返回值通过回调函数拿来的数据

在组件中使用：

在main.js中写

Import VueJsonp from ‘vue-jsonp’

Vue.use(VueJsonp)

在组件中使用 this.$jsonp(url,{参数}).then().catch()

在store中使用

Import VueJsonp from ‘vue-jsonp’

在actions里面写Vue.jsonp(url).then(res){}

分模块：

modules:{模块名称：模块对象的引用}

使用：this.$store.state.模块名称.变量

...mapState([‘模块名称’,’变量’])

动画：

v-enter v-enter-active v-enter-to

v-leave v-leave-active v-leave-to

mode: out-in 先进行过度动画再进入

项目开发流程：

1. 客户提出需求-->产品经理出需求文档
2. 产品出原型图
3. 做设计稿
4. 交该开发团队做开发
   1. 根据需求划分模块
   2. 项目估期和模块估期
   3. 分配责任人
   4. 根据分配的模块来确认后端接口，后端要出接口文档，前端要验收接口文档
      1. Url，methods,params,返回值
   5. 建立项目框架，上传到代码仓库，其他的开发人员要下载仓库里的代码进行开发，前端要进行模拟后端的接口（mock数据）
   6. 等后端开发完接口要进行前后端的联调（开发服务器）
   7. 测试 （测试服务器上进行测试）
   8. 安全团队
   9. 上线（生产服务器）