Vue is declarative (Define a template of part)

Const app = vue.createApp {data() return}去设置Vue, app.mount(#id)

如果Vue 控制某个HTML组件， 则该组件的所有子组件都被Vue控制.

**v-bind** -> Bind/Set attribute values

**v-html** -> values is HTML element

**v-on:click** -> Button Click event trigger

**methods** ALLOW function -> pass JS function.

Method:{ functionName() { function } }

Data are stored in global.

functionName(event) : this.attribute = event.target.value -> 读取input值

preventDefault -> do not submit automatically.

Prevent modifier: Prevent browser default

Stop modifier : stopping paragation.

v-on:click -> left, right and middle click.

v-once: Only be evaluated ONCE.

Keyboard modifier (keyup/ keydown)

v-model = v-bind:value + v-on:input

(**2 way binding** , listen the event and write the event)

DOM Interaction : (templates and data binding).

**Computed Properties**: once dependence array changes. Defined amount of methods but execute differently from methods.

For performance, computed properties are better than methods for **outputting values** in most cases.

Using Method to Re-calculate events otherwise using computed properties(FOR OUTPUT).

**Watch** (option) such as methods, data, computed property(similar):

Once the data property changes, the **watch** method will re-execute again.

**Watch**:{PropertyName(**value){method}**} -> when the data meet specific condition, then, it will changed by inside method.

**Watch** Option will help to set HTTP/ setTimer Method .

Using **Watch** Option when we need to add more functionality after **the condition meet**. Using **Computed** Option to **OUTPUT values.**

**Short Summary:**

* **Method:** Data Bind(executed for every re-render), Event Bind.
  + **Use for events/data that REALLY needs to be re-evaluated all time**
* **Computed: Data Bind,** ONLY re-evaluated if one of used values change
  + **(Use for data that depends on other data)**
* **Watch:** Not directly use in template,
  + **Use for any non-data update you want to make.**
* **V-on shorthand : @; v-bind shorthand : “:”**

**Inline CSS:**

* **:style=”{attribute: }”**

**v-if (show tag after meet conditions)**

**v-else (directive after v-if) ->** Showing content conditionally

**v-show(**v-show are more like Not shown, v-else more like NOT exist)

**v-for**(using “in” as keyword) -> v-for = “I in arrayName” (Such as for loop)

* Value in {name:’val1’, age:’val2’} -> read the value of property.
* (I, index) in arrayName ->{{i}}-{{index}} : Element – element position
* V-for=”(value,key,index) in {name:’val1’,age:’val2’}”
* V-for=”num in NUMBERS” -> get #
* Using **key attribute,** :key=”i” (unique identification)

Array -> Using **push** to **add** element into this element

* Using **splice**(index, position) -> find this index and **remove** from this array

**Stop** event modifier -> This is stop processing of previous functionality

“this” property not only using in **data** but also in **methods**.

Component(Custom HTML element) : 1. **Identifier** (contain a dash( ‘-‘)

* Template

Vue create [VueAppName] ->npm run server

In vue file: <template>(Contain HTML Element), <script>(Functionality),

Component-> app.component(ComponentName)

**Props:** parent child communication (**pass** data from parent to child)

* **Props** should NOT mutated
* Once, the data passed from parent to child. It can NOT changed by parent
* Set a new value in data, and set the value to this.propName.
* Props can set type(such as string), required, default, validator(function),
* Type: String , Number Boolean Array Object Date Function Symbol
* **Using v-bind to represent the value** -> v-for = element in array; key=”array.id” ; :name=”array.name”
* **This**.$emit() -> Allows to emit custom event , listen from parent component.

Filter : let arr = [1, 2, 3, 4, 5, 6, 7, 8];

let result = arr.filter((item) => item > 5);

console.log(result); // [ 6, 7, 8 ]

,reduce let arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

let result = arr.reduce(function (total, currentValue, currentIndex, arr) {

return total + currentValue;

}, 0);

console.log(result); // 55

, find let arr = [1, 2, 3, 4, 5, 6];

let result = arr.find((item) => item >= 2);

console.log(result); // 2

, map let arr = ["Tom", "Mike", "Shanguagua"];

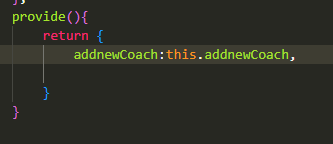
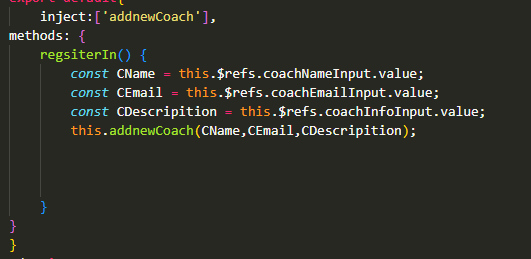
let result = arr.map((item, index) => {

return `第${index + 1}名：${item}`;

});

console.log(result); // [ '第1名：Tom', '第2名：Mike', '第3名：Shanguagua' ]

**Provide,Inject: (work together):**

* **Only Inject what has been provide**
* **如果想使用父类method在子类里面可以使用provide，inject**
* **父类：method 名称**
* ****
* **子类： 引用该method**
* ****

**Local vs Global Component:**

* Using **component** in export default, import ComponentName from location. Component:{ComponentName}
* If this **component** use in lots of other places, then We are going to use it globally(app.component() -> in main.js file)

**Styling**:

* If the style inside of vue file, then it is global styling.
* If **Style Scoped ->** Then it will only affect this file

**Slot:**

* Allow us to receive HTMl content from outside of the component.
* Using for HTML code.
* <slot name=”slotName”> to have more slots-> Sepific which slot need to be used in there. 和 <template v-slot:slotName>一起使用
* Slot 通俗的理解就是“占坑”，在组件模板中占好了位置，当使用该组件标签时候，组件标签里面的内容就会自动填坑（替换组件模板中slot位置）  
  并且可以作为承载分发内容的出口
* Slot will not change the props, component, methods….

Dynamic Component:

* <component :**is=”methodName”**></component>
  + 赋予component functionality
* <keep-alive> -> 不完全删除vue组件， 保存其状态与数据
* <teleport to=”attributeName(such as body)”>

**Form:**

* **@submit.prevent=”methodName”**
* **V-model(**Two way bind, not only listen to the event, but also can change the value) **来获取input的值. (vmodel = @input=, :value)**
  + **V-model.typeName(number,text) ->** 让vmodel来确定输入类型
  + **如果使用this.$ref.refName.value ->** 返回的是String
  + v-model 在select 上和input 有相同效果
  + v-model=”DataName” , Methods(){ this.DataName}获取input值
  + 如果有多个checkbox，并且name相同的情况下，v-model要使用array来储存，并且每个checkbox需要有value
  + 如果只有单一的checkbox，则使用false， true在vmodel
  + @blur 用于判断是否为焦点，或用于验证是否输入
    - 如果用户点击这个input，则触发blur
  + 对component使用vmodel， vmodel= :model-value=’’ , @update:modelValue=’’

在custom component里， props:['modelValue'],

    emits:['update:modelValue'],

    methods:{

        activate(*option*){

            this.activeOption = *option*;

            this.$emit('update:modelValue',*option*);

        }

    } 来获取Input Value

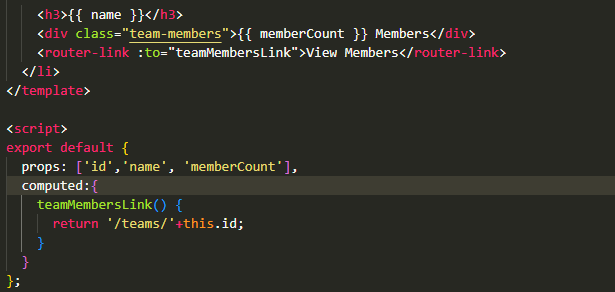
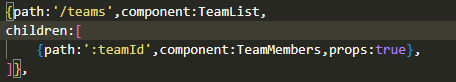
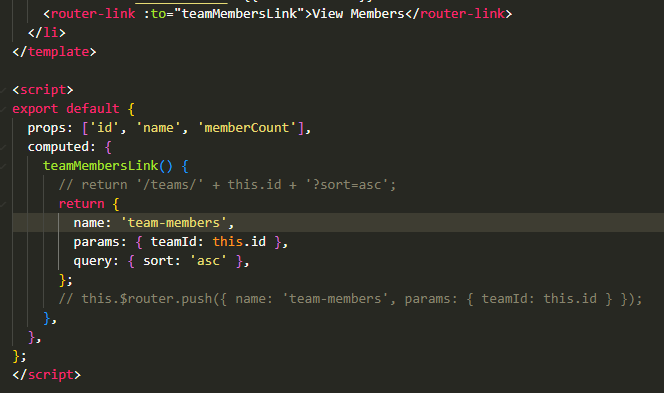
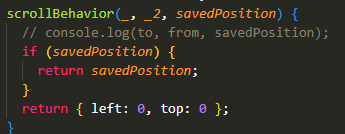
设置backend：

* Axios/fetch(‘url”) 用于send http request
* fetch('https://module12-f5f62-default-rtdb.firebaseio.com/survey.json' , {
* method: 'POST',
* headers:{
* 'Content-Type': 'application/json' 发送json
* },
* Body:JSON.stringify{name:this.userName,rating:this.chosenRating}
* });
* Fetch return object, which can listen.once the data in there, it will execute
* Fetch(‘url).then(function Name() { if(response.ok){response.json();} }).then(function(data) {console.log(data}).catch((error)=>{this.error=’text’ });
  + 在http请求完成后，用then触发GET 数据
  + Lesson158
  + Catch 为报错后的方案
* LoadingMessage:false,
  + 在fetch之前set为true， GET数据后，set为false
  + Lesson159
* 如果data form的length <0用于判断是否要显示no data
* Error Response
  + Lesson162

**VueRouter: (import createRouter from vue-router) -> const router=createRouter(object)**

* Create an **oject** inside of createRouter,
  + Object contain: **history(如果transfer到其他页面， 该package可使他回到原来的界面), routes:[],**
  + History: **createwebhistory()**
  + Routes: which component should been mounted to which URL
  + **Routes:[ { path : ‘/pathName’,componentName(import VueFile)} ]**
    - **当url为当前path，则load指定component**
  + App.use(router)
  + Use <**router-link to=”/pathName”>** inside of each component to transfer page and update the content.
    - Similar to anchor tag <a>
    - A.router-link-active -> CSS渲染 (在该anchor tag被触发后)
  + **This.$router.push/back/forward(‘/path’) ->** 在event handler触发后， 使用该方法转到其他页面
  + **{path:/teams/:Dynamic , component: ComponentName}**
  + **如果有dynamic和static path， 则需要把static放在dynamic前面**
  + **根据动态id来显示value**

****

* + 根据点击的button来显示不同的page value
  + 
* Redirect -> 直接转到 一个page
* 
* 如果有无效链接
* 
* Path扩展（dynamic link） 使用children (**nested router)**
* 
* GET 更多信息关于 URL使用**QUERY**
* 
* 可以拥有多个router view， 用《router-view **name=”routername”>**来区分.
  + **在routerJS里面**
* 每次当页面改变的时候， 使用**scrollbehavior（to,from,savedposition）来转到相应的页面位置**
  + **返回一个值（object）， 网页应该滚动到哪个位置当改变的时候.savedposition 为该值**
  + 
* Router.**beforeEach**(function()) -> 使用function为参数
  + 当router 转到另外的页面是运行该function
  + Function包含三个参数 **to,from,next**
    - Next() /next(true)-> 允许切换页面
    - Next(false) ->拒绝切换页面
* beforeEnter(to,from,next){}
* beofreRouteEnter -> next
* **beforeEach(global)>** beforeEnter> beofreRouteEnter(component)
* **beforeRouteUpdate（to,from,next）**-> 在UPDATE之前检查是否允许navigate
* **afterEach(function(to,from)) -> 没有next**
  + **发送数据/获取用户数据**

**CSS (Transition and Animation) in Vue**

* Transform:translateX（） -> 在X坐标轴上移动多少个px
* Transition: opacity/all/transform 运行多少时间 ， ease-in/out
  + 当以上attribute被调用时，自动设置为动画模式， 运行多少秒， east in/out
* **@keyframe Name**{

**0%{**

**Transform:translateX() scale(1)**

* + - **Scale是增大、减小该obj的一个函数**

**}**

**50%{**

**Functionality**

**}**

**0，50，70 是动画进行的进度**

}

* ClassName {

Animation: Name(@keyframe) seconds ease-in/out

} 来使用keyframe

* @Keyframe Name{

**From{**

**Opacity:0**

**Transform:translateX, scale**

**}**

**To{**

**Opacity:1**

**Transform: translateY() scale**

**}**

}

对话框弹窗

Animation: name second ease-out forward;

* **<transition> Vue tag**
  + **In style tag:**
    - .v-enter-from{opacity, transform:}
    - .v-enter-active{transition:all second}
    - .v-enter-to{opacity, transform:}
    - Enter-from, enter-to: 初始位置和预计展现的位置，
    - Enter-active是正在展示的状态，
    - .v-leave-from
    - .v-leave-active
    - .v-leave-to{可以使用和enter-from一样的内容}
  + className-Enter-from, className-enter-active,

className-enter-to in <transition name=’className’>

* + - **transition 必须有一个direct 子元素**
  + **<transition mode=””>**
    - **In-out (in and remove)**
    - **Out-in (remove and genereate in new)**
  + Using v-if, v-else 来使用多个direct子元素
* JavaScript control Animation
  + GreenSock
  + In <transition

name,

@before-enter=”methods”

@Enter=”methods”

@After-enter=”methods”

@Before-leave=”methods”

@Leave=”methods”

@After-leave=”methods”

@enter-cancelled=”methods”

@leave-cancelled =”methods” >

* + 这些@methods ->都设置在methods 里面
  + **Let round=1;**
  + **Enter(el,done)** (Const this.enter(leave)interval = setInterval(function(name.style.opacity= round\*0.1 ;round++;每1秒透明度增加10%，

if(round>10)clearInterval(interval);done() 如果round>10, 则清空)并结束, time))

**-> setInterval(function(),second)允许运行代码每几毫秒**

* + **Leave(el,done) ->** 和enter相同逻辑， 只是数值要相反
  + enter-cancelled, leave-cancelled(clearInterval(this.enter(leave)interval)
  + enter(leave)interval=null
* 获取input值 const Name = this.$ref.refName.value;
* Unshift() 放置到top，
* <Transition-group>:
  + **加入多个动画**
  + **比如<transition-group tag=’ul’ name=’name’>**
    - **<li v-for=’I in array’ :key=I @click=’methods’>**
    - Style里面：.name-enter-from{动画的functionality},

.name-enter-active{**transition:all second, ease-in/out** }, .name-enter-to

* **Router Animation**
  + <router-view>

<transition name=’name’ mode=”out-in”>

**<component :is=’slotProps.Component’ />**

</transition>

</router-view>

* + In App.vue
    - .name-enter-from
    - .name-enter-active
    - .name-enter-to
    - (opacity, transltaeX,translateY) 来切换页面
  + Router.isready().then(function(){

**App.mount(#app)**

});

是否加载成功

如果加载成功app.mount

* **VueX(**Manage global state)
  + **作用**
    - **Outsourced state Management**
    - **Predictable state management/flow**
    - **Clear defined data flow**
  + Import **createStore from vuex**

*const* store=createStore({

    state(){

        return {

            counter:0,

        };

    }

}) app.use(store)

//在任何component里面皆可使用该store property

$store.state.counter -> 显示了全局储存的值

* + **Mutations**
    - Trigger mutations INSTEAD OF control the state
    - 接受**object** ， 内部储存function
    - 使用$this.StoreName.commit(state,payload)
* mutations:{Mutation 在mainJS里面 是createStore的一个内置methods
* increment(*state*){
* *state*.counter++;
* }
* }
* addup(){
* this.$store.commit(MutationName);
* 在component里面使用该函数来触发mutation
* }

在mutations里面传入第二个值，可以自定义这个value的值

* increase(*state*,*payload*){
* *state*.counter += *payload*.value;
* }
* this.$store.commit('increase',{value:20});
  + **Getters (**更好的方式获取数据)
    - 接受**object** ， 内部储存function. **getter{function()}**
    - Getters 可以互相依赖
    - This.$store.getter.MethodName(该方法储存在mainJS，getter下面)
  + **Actions**
    - 允许使用async代码
    - Actions:{actionName(context) { setTimeout(function(**context.commit(‘actionName)**),time } }
    - Function里面可以包含 state，payload
    - 在component里面：
      * This.$store.dispatch(actionName, Payload)
      * This.$store.dispatch(actionName)
    - **Context.commit/dispatch/getters/state**
  + **mapGetters**
* import {mapGetters} from vuex
* computed:{
* ...mapGetters(['multiple'])
* }
* 可以使用该方法存储多个getters在一起，在**computed方法里**
* 在mapgetters存储的method会直接指向mainJS里的getter内置method
* import {mapActions} from vuex
* methods:{
* ... mapActions ({mul: 'multiple'})
* }
* 可以使用该方法存储多个actions在一起，在**methods方法里**
* 在mapActions存储的method会直接指向mainJS里的actions内置method
  + **Module:**
    - **如果有两个以上的createStore attribute**
    - **则可以使用module来作为其merge的选项**
* modules: {
* numbers: counterModule
* },
  + - **用numbers作为整个counterModule（redux）的索引词**