四、Vue.js

1. V-if和V-show的区别

v-if：动态为元素添加删除标签元素来切换显示隐藏，有较高的切换渲染，是真正的条件渲染，可结合v-else-if和v-else等指令一起使用。

v-show：切换元素的display属性来切换元素的显示与隐藏。有较高的初始渲染。

1. vue生命周期以及钩子函数

总共分为8个阶段创建前/后，载入前/后，更新前/后，销毁前/后。

创建前/后： beforeCreated阶段，vue实例的挂载元素$el和data都为undefined，还未初始化。在created阶段，vue实例的数据对象data初始化了，$el还没有。

载入前/后：在beforeMount阶段，vue实例的$el和data都初始化了，但还是挂载之前为虚拟的dom节点，data.message还未替换。在mounted阶段，vue实例挂载完成，data.message成功渲染。

更新前/后：当data变化时，会触发beforeUpdate和updated方法。

销毁前/后：在执行destroy方法后，对data的改变不会再触发周期函数，说明此时vue实例已经解除了事件监听以及和dom的绑定，但是dom结构依然存在

1. vue如何实现数据的双向绑定

VueJS使用 ES5提供的 Object.defineProperty方法，监控对数据的操作，从而可以自动触发数据同步。并且，由于是在不同的数据上触发同步，可以精确的将变更发送给绑定的视图，而不是对所有的数据都执行一次检测。

vue.js 是采用数据劫持结合发布者-订阅者模式的方式，通过Object.defineProperty()来劫持各个属性的setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调。

具体步骤：

第一步：需要observe的数据对象进行递归遍历，包括子属性对象的属性，都加上 setter和getter；这样给这个对象的某个值赋值，就会触发setter，那么就能监听到数据变化。

第二步：compile解析模板指令，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图。

第三步：Watcher订阅者是Observer和Compile之间通信的桥梁，主要做的事情是:

1、在自身实例化时往属性订阅器(dep)里面添加自己

2、自身必须有一个update()方法

3、待属性变动dep.notice()通知时，能调用自身的update()方法，并触发Compile中绑定的回调，则功成身退。

第四步：MVVM作为数据绑定的入口，整合Observer、Compile和Watcher三者，通过Observer来监听自己的model数据变化，通过Compile来解析编译模板指令，最终利用Watcher搭起Observer和Compile之间的通信桥梁，达到数据变化 -> 视图更新；视图交互变化(input) -> 数据model变更的双向绑定效果。

Object.defineProperty(obj, prop, descriptor): 直接在一个对象上定义一个新属性，或者修改一个对象的现有属性， 并返回这个对象。

obj：要在其上定义属性的对象。

prop：要定义或修改的属性的名称。

descriptor：将被定义或修改的属性描述符。

属性描述符：数据描述符和存取描述符。描述符必须是这两种形式之一；不能同时是两者。

公共属性：configurable：当且仅当该属性的 configurable 为 true 时，该属性描述符才能够被改变，同时该属性也能从对应的对象上被删除。默认为 false。

enumerable：当且仅当该属性的enumerable为true时，该属性才能够出现在对象的枚举属性中。默认为 false。

数据描述符是一个具有值的属性，该值可能是可写的，也可能不是可写的。

value：该属性对应的值。可以是任何有效的 JavaScript 值（数值，对象，函数等）。默认为 undefined。

writable：当且仅当该属性的writable为true时，value才能被赋值运算符改变。默认为 false。

存取描述符是由getter-setter函数对描述的属性。

get：一个给属性提供 getter 的方法，如果没有 getter 则为 undefined。当访问该属性时，该方法会被执行，方法执行时没有参数传入，但是会传入this对象（由于继承关系，这里的this并不一定是定义该属性的对象）。默认为 undefined。

set：一个给属性提供 setter 的方法，如果没有 setter 则为 undefined。当属性值修改时，触发执行该方法。该方法将接受唯一参数，即该属性新的参数值。默认为 undefined。

Object.defineProperty(obj, "b", {

get : function(){

return bValue;

},

set : function(newValue){

bValue = newValue;

},

enumerable : true,

configurable : true

});

1. vue父子组件之间传值

父组件向子组件传值：$ref； props：注重数据传递

子组件向父组件传值：$emit()

1. 非父子组件之间的传值：
   1. 、bus总线机制

const bus = new Vue() // bus也是一个vue实例对象

②、使用vuex状态管理器

1. Vue-loader：

Vue-loader是基于webpack解析.vue文件的一个加载器loader，解析和转换 .vue 文件，提取出其中的逻辑代码 script、样式代码 style、以及 HTML 模版 template，再分别把它们交给对应的 Loader 去处理。最后将他们组装成一个commonjs模块；module.exports出一个vue.js组件对象。

vue-loader 的作用：

1. 允许为 Vue 组件的每个部分使用其它的 webpack loader，例如在 <style> 的部分使用 Sass 和在 <template> 的部分使用 Pug；
2. 允许在一个 .vue 文件中使用自定义块，并对其运用自定义的 loader 链；
3. 使用 webpack loader 将 <style> 和 <template> 中引用的资源当作模块依赖来处理；
4. 为每个组件模拟出 scoped CSS；
5. 在开发过程中使用热重载来保持状态。
6. Vue中的slot插槽使用

slot：父组件需要在子组件内放一些DOM，那么这些DOM是显示、不显示、在哪个地方显示、如何显示，就是slot分发负责的活。插槽显不显示、怎样显示是由父组件来控制的，而插槽在哪里显示就由子组件来进行控制。

插槽用于将所携带的内容，插入到指定的某个位置，从而使模板分块，具有模块化的特质和更大的重用性。

$destroy()销毁组件，没有销毁slot。

单个插槽：当子组件模板只有一个没有属性的插槽时，父组件传入的整个内容片段将插入到插槽所在的 DOM 位置，并替换掉插槽标签本身。

最初在 <slot> 标签中的任何内容都被视为备用内容。备用内容在子组件的作用域内编译，并且只有在宿主元素为空，且没有要插入的内容时才显示备用内容。

具名插槽：solt元素可以用一个特殊的特性name来进一步配置如何分发内容。多个插槽可以有不同的名字。  
这样可以将父组件模板中 slot 位置，和子组件 slot 元素产生关联，便于插槽内容对应传递。

作用域插槽(scoped slots)：可以访问组件内部数据的可复用插槽(reusable slot)  
在父级中，具有特殊特性 slot-scope 的<template> 元素必须存在，表示它是作用域插槽的模板。slot-scope 的值将被用作一个临时变量名，此变量接收从子组件传递过来的 prop 对象。

1. 计算属性computed和 watch 的区别:

计算属性是依赖的值改变会重新执行函数，计算属性是取返回值作为最新结果，所以里面不能异步的返回结果。不能写异步逻辑。

侦听属性是侦听的值改变会重新执行函数，将一个值重新赋值作为最新结果，所以赋值的时候可以进行一些异步操作。

所以区别来源于用法，只是需要动态值，那就用计算属性；需要知道值的改变后执行业务逻辑，才用 watch，用反或混用虽然可行，但都是不正确的用法。

computed 是一个对象时，它有哪些选项？  
computed 和 methods 有什么区别？  
computed 是否能依赖其它组件的数据？  
watch 是一个对象时，它有哪些选项？

1. 有get和set两个选项
2. methods是一个方法，它可以接受参数，而computed不能，computed是可以缓存的，methods不会。
3. computed可以依赖其他computed，甚至是其他组件的data
4. watch 配置  
   handler  
   deep 是否深度  
   immeditate 是否立即执行

应用场景:

watch：侦听一个数的变化，当该数据变化时，来处理其他与之相关数据的变化（该数据影响别的多个数据）

computed：获得一个值或结果，该结果受其他依赖的影响。（一个数据受多个数据影响）

1. vue-router路由实现原理

router-link：页面中相当于a链接的标签，to跳转时url中不用加#

router-view：显示路由配置path对应的组件内容

实现路由的方式有两种：vue通过参数mode来设置，默认是hash模式。

hash模式：

使用 URL 的 hash 来模拟一个完整的 URL，于是当 URL 改变时，页面不会重新加载。

每次hash值的变化会触发hashchange这个事件，通过这个事件就可以知道 hash 值发生了哪些变化，然后便可以监听hashchange来实现更新页面部分内容的操作。

history模式：单页路由的 url 就不会多出一个#，变得更加美观。

这种模式充分利用 history.pushState API 来完成 URL 跳转而无须重新加载页面。

当使用 history 模式时，URL 就像正常的 url，例如 http://yoursite.com/user/id

不过这种模式还需要后台配置支持。因为我们的应用是单页客户端应用，如果后台没有正确的配置，当用户在浏览器直接访问 http://oursite.com/user/id 就会返回 404

所以要在服务端增加一个覆盖所有情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回同一个 index.html 页面，这个页面就是app 依赖的页面。

history.pushState()与hash比较：

1. pushState设置的新url可以是与当前url同源的任意url，而hash只可修改#后面的部分，故可设置与当前同文档的url
2. pushState设置的新url可以与当前url一模一样，这样也会把记录添加到栈中，而hash设置的新的值必须与原来不一样才会触发记录添加到栈中
3. pushState通过stateObject可以添加任意类型的数据记录中，而hash只可添加短字符串
4. pushState可额外设置title属性供后续使用。
5. 动态路由匹配：

需要把某种模式匹配到的所有路由，全都映射到同个组件。使用 “动态路径参数”来实现。

一个“路径参数”使用冒号 : 标记。当匹配到一个路由时，参数值会被设置到 this.$route.params，可以在每个组件内使用。

定义动态路由：在router目录下的index.js文件中，对path属性加上…/:\*\*\*。

路由组件传参：

组件中获取动态路由参数：使用router对象的params.id：$route.params.\*\*\*

但是在组件中直接使用 this.$route.params 会降低组件复用的灵活性。

更好的办法是将变量以props的形式传递给组件。 也更加方便组件的重用和测试。

有3种在路由配置中给组件传递props的模式：布尔模式，对象模式，函数模式

1. vue-router如何响应路由参数的变化:

当使用动态路由参数时，例如从 /content?id=1 到 content?id=2，此时原来的组件实例会被复用。这也意味着组件的生命周期钩子不会再被调用，此时vue应该如何响应路由参数的变化？

方法一：使用watch的$route对象

watch: { '$route' (to, from) { // 对路由变化作出响应... } }

//路径发生变化，$route会重新赋值，监控了这个属性，会执行函数内代码

方法二：引入beforeRouteUpdate守卫：beforeRouteUpdate (to, from, next){ }

1. vue-router导航钩子函数

“导航”表示路由正在发生改变。

vue-router 提供的导航守卫主要用来通过跳转或取消的方式守卫导航。有多种机会植入路由导航过程中。参数或查询的改变并不会触发进入/离开的导航守卫。

导航钩子有：

全局导航钩子：前置守卫（beforeEach）、解析守卫（beforeResolve）、后置钩子（afterEach）

前置守卫：router.beforeEach ( (to, from, next) => { // ... } );

每个守卫方法接收三个参数：

to: Route:即将要进入的目标 路由对象

from: Route: 当前导航正要离开的路由

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。确保要调用 next方法，否则钩子就不会被 resolved，如果不用就拦截

* next(): 进行管道中的下一个钩子。如果全部钩子执行完了，则导航的状态就是confirmed (确认的)。
* next(false): 中断当前的导航。如果浏览器的 URL 改变了 (可能是用户手动或者浏览器后退按钮)，那么 URL 地址会重置到 from 路由对应的地址。
* next(’/’) 或者 next({ path: ‘/’ }): 跳转到一个不同的地址。当前的导航被中断，然后进行一个新的导航。你可以向 next 传递任意位置对象，且允许设置诸如 replace: true、name: ‘home’ 之类的选项以及任何用在router-link的 to prop或 router.push中的选项。
* next(error): (2.4.0+) 如果传入 next 的参数是一个 Error 实例，则导航会被终止且该错误会被传递给 router.onError()注册过的回调。

解析守卫：在导航被确认之前，同时在所有组件内守卫和异步路由组件被解析之后，解析守卫就被调用。

全局后置钩子：和守卫不同的是，这些钩子不会接受 next 函数也不会改变导航本身。

router.afterEach((to, from) => { // ... });

路由配置的守卫：beforeEnter: (to, from, next) => { // ... }

组件内的导航钩子：beforeRouteEnter、beforeRouterUpdate、beforeRouteLeave 、afterEnter

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` ；因为当守卫执行前，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 之间跳转的时候；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 (2.2+)。
5. 在路由配置里调用 beforeEnter。
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 (2.5+)。
9. 导航被确认。
10. 调用全局的 afterEach 钩子。
11. 触发 DOM 更新。
12. 用创建好的实例调用 beforeRouteEnter 守卫中传给 next 的回调函数。
13. nextTick()：

在下次 DOM 更新循环结束之后执行延迟回调。在修改数据之后，立即使用这个回调函数，获取更新后的DOM。

两种写法：全局：Vue.nextTick(callback)； 局部：vm/this.$nextTick(callback)

vue中数据和dom渲染由于是异步的，所以要让dom结构随数据改变这样的操作都应该放进this.$nextTick()的回调函数中。

应用场景：

1.在Vue生命周期的created()钩子函数进行的DOM操作一定要放在Vue.nextTick()的回调函数中，因为在created()钩子函数执行的时候DOM并未进行任何渲染，此时进行DOM操作无异于徒劳，所以一定要将DOM操作的js代码放进Vue.nextTick()的回调函数中。这样就相当于是在mouted里调用这个生命钩子， mounted钩子函数执行时所有的DOM挂载和渲染都已完成，此时在该钩子函数中进行任何DOM操作都不会有问题 。

2.在数据变化后要执行的某个操作，而这个操作需要使用随数据改变而改变的DOM结构的时候，这个操作都应该放进Vue.nextTick()的回调函数中。在下次 DOM 更新循环结束之后执行延迟回调。在修改数据之后立即使用这个方法，获取更新后的 DOM。

注意⚠： mounted 不会承诺所有的子组件也都一起被挂载。如果希望等到整个视图都渲染完毕，可以用 vm.$nextTick 替换掉 mounted

1. 对Vue.js的template编译的理解：

简述：先转化成AST树，再得到的render函数返回VNode（Vue的虚拟DOM节点）。

AST：抽象语法树，是源代码的抽象语法结构的树状表现形式，这里特指编程语言的源代码。

createCompiler：用以创建编译器，返回值是compile以及compileToFunctions。

Compile：是一个编译器，会将传入的template转换成对应的AST树、render函数以及staticRenderFns函数。另外compile还负责合并option（createCompiler会根据平台区分传入一个baseOptions，compile会将平台自有的option与本身传入的option进行合并得到最终的finalOptions）。

首先，通过compile编译器把template编译成AST语法树；

然后，AST会经过generate（将AST语法树转化成render funtion字符串的过程）得到render函数，render的返回值是VNode，VNode是Vue的虚拟DOM节点，里面有（标签名、子节点、文本等等）。

1. Keep-alive：

keep-alive是Vue提供的一个抽象组件。它自身不会被渲染成一个DOM元素，也不会出现在父组件链中；使用keep-alive包裹动态组件时，会缓存不活动的组件实例，而不是销毁它们，避免了组件反复创建和渲染，有效提升系统性能。总的来说，keep-alive用于保存组件的渲染状态。

当组件在keep-alive内被切换时组件的activated、deactivated这两个生命周期钩子函数会被执行。这两个钩子函数只在使用keep-alive之后才会有，否则不会存在。

activated()：被包裹组件被激活时调用，在vue对象存活的情况下，进入当前存在activated()函数的页面时，一进入页面就触发；可用于初始化页面数据等。

deactivated()：被包裹组件被移除时调用

被包裹在keep-alive中的组件的状态将会被保留，例如我们将某个列表类组件内容滑动到第100条位置，那么我们在切换到一个组件后再次切换回到该组件，该组件的位置状态依旧会保持在第100条列表处。

用法：

<keep-alive :include="whiteList" :exclude="blackList" :max="amount">

<component :is="currentComponent"></component>

或者<router-view></router-view>

</keep-alive>

include定义缓存白名单，keep-alive会缓存命中的组件；exclude定义缓存黑名单，被命中的组件将不会被缓存；max定义缓存组件上限，超出上限使用LRU的策略置换缓存数据。

1. vuex相关



vuex：vue框架中状态管理模式。

应用场景：中大型单页应用中，多个组件共享数据或者是跨组件之间需要传递数据.通过状态（数据源）集中管理驱动组件的变化。

Vuex的状态和属性：

有五种状态： State、 Getter、Mutation 、Action、 Module （就是mapAction等）

vuex的流程：

当组件进行数据修改的时候需要调用dispatch来触发actions里面的方法。actions里面的每个方法中都会有一个commit方法，当方法执行的时候会通过commit来触发mutations里面的方法进行数据的修改。mutations里面的每个函数都会有一个state参数，这样就可以在mutations里面进行state的数据修改，当数据修改完毕后，会传导给页面。页面的数据也会发生改变。

特性：

Vuex 的状态存储是响应式的。当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新。

不能直接改变 store 中的状态。改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation。这样使得我们可以方便地跟踪每一个状态的变化，从而让我们能够实现一些工具帮助我们更好地了解我们的应用。

state：就是存放数据/状态的地方，类似一个容器仓库

action：包含任意异步操作，通过提交 mutation 间接更变状态而不是直接变更状态。

Action通过 store.dispatch()方法分发action中的方法，action中的commit可以触发mutations中的方法.若是同步，则直接在组件中commit触发vuex中mutations的方法。

mutation： 必须是同步函数, 更改 store 中的状态的唯一方法是提交 mutation。

Vuex 中的 mutation 非常类似于事件：每个 mutation 都有一个字符串的 事件类型 (type) 和一个回调函数。这个回调函数就是实际进行状态更改的地方，并且它会接受 state 作为第一个参数：store.commit('increment')

Getter：可以认为是 store 的计算属性。就像计算属性一样，getter 的返回值会根据它的依赖被缓存起来，且只有当它的依赖值发生了改变才会被重新计算。

Module: 将 store 分割成模块（module）。每个模块拥有自己的 state、mutation、action、getter、甚至是嵌套子模块——从上至下进行同样方式的分割.

vuex和本地存储的区别:

vuex 可以实时改变其他依赖状态值的状态，本地存储不能.

刷新页面之后vuex数据消失，本地存储不会

1. axios相关

axios：是一个请求后台资源的模块。返回一个promise对象。axios.get() / post()发送请求

axios.post(‘api/user’).then()：跨域添加用户操作；

axios.put(‘api/user/8′)：跨域更新操作

1. vue.cli项目中src目录每个文件夹和文件的用法

assets文件夹是放静态资源；components是放组件；router是定义路由相关的配置;view视图；app.vue是一个应用主组件；main.js是入口文件