前端-VUE面试题

笔记本:	前端知识点		
创建时间:	2020/8/15 15:22	**更新时间:**	2020/9/6 20:33
作者:	1317468898@qq.com		
URL:	https://www.cnblogs.com/fundebug/p/10454641.html		

1. 谈谈你对MVVM开发模式的理解

MVVM分为Model、View、ViewModel三者。

- **Model** 代表数据模型，数据和业务逻辑都在Model层中定义；
- **View** 代表UI视图，负责数据的展示；
- **ViewModel** 负责监听 **Model** 中数据的改变并且控制视图的更新，处理用户的交互操作；

Model 和 **View** 并无直接关联，而是通过 **ViewModel** 来进行联系的，
Model 和 **ViewModel** 之间有着双向数据绑定的联系。因此当 **Model** 中的数据改变时会触发 **View** 层的刷新，**View** 中由于用户交互操作而改变的数据也会在 **Model** 中同步。

这种模式实现了 **Model** 和 **View** 的数据自动同步，因此开发者只需要专注对数据的维护操作即可，而不需要自己操作 **dom**。

2. Vue 有哪些指令，如何实现?

v-html、v-show、v-if、v-for、v-model等等

2. Vue如何自定义指令?

使用 Vue.directive(id, [definition]) 定义全局的指令来进行自定义指令

参数1 ： 指令的名称，在定义的时候，指令的名称前面，不需要加 v-前缀，但是：在调用的时候，必须在置顶的名称前加上 v-前缀来进行调用

参数2： 是一个对象， 这个对象身上，有一些指令相关的函数，这些函数可以在特定的阶段，执行相关的操作。

```
全局指令：Vue.directive("focus", {
// 注意：在每个函数中， 第一个参数永远是el， 表示被绑定了指令的那个元素，这个el参数，是一个原生的JS对象
bind: function(el){ // 每当指令绑定到元素上的时候，会立即执行这个bind函数，只执行一次
},
inserted: function(el){ // inserted 表示元素插入到DOM中的时候，会执行inserted函数【触发一次】
el.focus()
},
updated: function(el) { // 当VNode更新的时候，会执行updated，可能会触发多次
```

```
},
})
局部指令：//创建根实例
new Vue({
    el: '#app',
    directives: {
        //注册一个局部的自定义指令 v-focus
        focus: {
            //指令的定义
            bind: function(el){ // 每当指令绑定到元素上的时候，会立即执行这个bind函数，只执行一次},
            inserted: function(el){ // inserted 表示元素插入到DOM中的时候，会执行inserted函数【触发一次】
                el.focus()
            },
            updated: function(el) { // 当VNode更新的时候，会执行updated，可能会触发多次},
        }
    }
})
```

调用：

```
<!-- 注意： Vue中所有的指令，在调用的时候，都以 v- 开头 -->
<input type="text" class="form-control" v-model="keywords" v-focus>
```

3. v-if 和 v-show 有什么区别?

v-show 仅仅是CSS切换；而v-if是一个完整的销毁与重建。当我们需要经常切换某个元素的显示/隐藏时，使用v-show会更加节省性能；当只需要一次显示或隐藏时，使用v-if更加合理。

3. 事件处理器：v-on

- 事件修饰符： .prevent(阻止默认事件，比如提交事件不会重载页面)、.stop(阻止冒泡)、.capture(阻止捕获事件)、.self(只监听触发该元素的事件，只当event.target在元素本身时触发时触发回调，非子元素)、.once(只触发一次)、.left(左键)、.right(右键)、.middle(中间滚轮)
- 按键修饰符：.enter、.delete、.table、.up、.down
- 修饰符可以串联，@click.prevent.self会阻止所有的点击事件；@click.self.prevent只会阻止对元素自身的点击；

4. 简述Vue的响应式原理

在生成一个vue实例时，对传入的data进行遍历，用 **Object.defineProperty** 将它们转为getter/setter，并且在内部追踪相关依赖，在属性被访问和修改时通知变化。

每个vue实例都有相应的 watcher 程实例，它会在实例渲染时记录这些属性，并在setter 触发时重新渲染。

4. computed vs methods

可以使用methods来替代computed，效果上两个都是一样的，但是computed是基于它的依赖缓存，只有相关依赖发生改变时才会重新取值。而使用methods，在重新渲染的时候，函数总会

重新调用执行。

使用 computed 性能会更好，但是如果不希望缓存，可以使用 methods 属性。

4. computed vs watch

computed （计算属性，依赖其他属性值，其值有缓存，只有它依赖的属性值发生改变，下次获取computed 的值时才会重新 computed 的值）

当我们需要进行数值计算，并且依赖于其它数据时，应该使用 computed ，因为可以利用 computed 的缓存特性，避免每次获取值时，都要重新计算；

watch （观察的作用，每当监听的数据变化时都会执行回调进行后续操作）

当我们需要在数据变化时执行异步或开销较大的操作时，应该使用 watch，使用 watch 选项允许我们执行异步操作（访问 一个 API），限制我们执行该操作的频率，并在我们得到最终结果前，设置中间状态。这些都是计算属性无法做到的。

5. 双向绑定原理

Vue2是采用数据劫持结合发布/订阅模式的方式，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。

首先我们为每个vue属性用Object.defineProperty()实现数据劫持，为每个属性分配一个订阅者集合的管理数组dep；

然后在编译的时候在该属性的数组dep中添加订阅者，v-model会添加一个订阅者，{{}}也会，v-bind也会，只要用到该属性的指令理论上都会；

接着为input添加监听事件，修改值就等于为该属性赋值，则会触发该属性的set方法，在set方法内通知订阅者数组dep，订阅者数组循环调用各订阅者的update方法更新视图。

Vue3是用ES6中的proxy实现的， 即在目标对象之前进行拦截，访问该对象属性需要先过拦截这一步骤。因此提供了一种机制，可以对外界的访问进行过滤和读写。

优缺点:

1.object.defineProperty无法监控到数组下标的变化，导致直接通过数组的下标给数组设置值，不能实时响应，而proxy可以检测到数组内部数据的变化。

2.Object.defineProperty只能劫持对象的属性,因此我们需要对每个对象的每个属性进行遍历。proxy可以劫持整个对象，并返回一个新对象，而且有13中劫持操作。

6. Vue中如何在组件内部实现一个双向数据绑定?

假设有一个输入框组件，用户输入时，同步父组件页面中的数据

具体思路：父组件通过 props 传值给子组件，子组件通过 $emit 来通知父组件修改相应的 props值，具体实现如下：

// 父组件

```
<aa-input v-model="aa"></aa-input>
// 等价于
<aa-input v-bind:value="aa" v-on:input="aa=$event.target.value"></aa-input>
// 子组件:
<input v-bind:value="aa" v-on:input="onmessage"></aa-input>
props:{value:aa,}
methods:{
   onmessage(e){
      $emit('input',e.target.value)
   }
}
```

6. v-model是如何实现的?

v-model 指令在表单控件元素上创建双向数据绑定，比如在input 上绑定value值，并通过input事件获取当前事件的target.value，并赋值给value。

<input type="text" :value="message" @input="message = $event.target.value">

7. 生命周期

beforeCreate （组件实例被创建之初，组件的属性生效之前；vue实例的挂载元素el和数据对象data都为undefined，还未初始化）

created（组件实例已经完全创建，属性也绑定，但真实 dom 还没有生成，$el 还不可用）

beforeMount （在挂载开始之前被调用：相关的 render 函数首次被调用；vue实例的$el和data都初始化了，但还是挂载之前未虚拟的DOM节点，data尚未替换）

mounted （el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子；vue实例挂载完成，data成功渲染）

beforeUpdate （组件数据更新之前调用，发生在虚拟 DOM 打补丁之前，不推荐使用）

update （组件数据更新之后，不推荐使用）

activited（keep-alive 专属，组件被激活时调用）

deactivated（keep-alive 专属，组件被销毁时调用）

beforeDestory （组件销毁前调用，通过removeEventListener解除手动绑定的事件）

destoryed （组件销毁后调用）

8. delete和Vue.delete删除数组的区别

delete只是被删除的元素变成了 empty/undefined ，其他的元素的键值还是不变。

Vue.delete直接删除了数组，改变了数组的键值。

9. 如何优化SPA应用的首屏加载速度慢的问题?

- 将公用的JS库通过script标签外部引入，减小app.bundel的大小，让浏览器并行下载资源文件，提高下载速度；
- 在配置 路由时，页面和组件使用懒加载的方式引入，进一步缩小 app.bundel 的体积，在调用某个组件时再加载对应的js文件；
- 加一个首屏 loading 图，提升用户体验

17. 虚拟DOM实现原理

用 JavaScript 对象模拟真实 DOM 树，对真实 DOM 进行抽象；

diff 算法 — 比较两棵虚拟 DOM 树的差异；

pach 算法 — 将两个虚拟 DOM 对象的差异应用到真正的 DOM 树。

10. 组件通讯/Vue组件的传参方式

props / $emit（适用 父子组件通信）

父组件通过props的方式向子组件传递，子组件通过本身的$emit和父组件的v-on通知父组件修改props值。

ref与$parent / $children（适用 父子组件通信）

需要注意的是：这两种都是直接得到组件实例，使用后可以直接调用组件的方法或访问数据

- ref：如果在普通的 DOM 元素上使用，引用指向的就是 DOM 元素；如果用在子组件上，引用就指向组件实例
- $parent / $children：访问父 / 子实例

$attrs/$listeners（适用于 隔代组件通信）

多级组件嵌套需要传递数据时，通常使用的方法是通过vuex。但如果仅仅是传递数据，而不做中间处理，使用 vuex 处理，未免有点大材小用。为此Vue2.4版本提供了另一种方法----$attrs/$listeners

- $attrs：包含了父作用域中不被 prop 所识别 (且获取) 的特性绑定 (class 和 style 除外)。当一个组件没有声明任何 prop 时，这里会包含所有父作用域的绑定 (class 和 style 除外)，并且可以通过 v-bind="$attrs" 传入内部组件。通常配合 interitAttrs 选项一起使用。
- $listeners：包含了父作用域中的 (不含 .native 修饰器的) v-on 事件监听器。它可以通过 v-on="$listeners" 传入内部组件

provide / inject（适用于 隔代组件通信）

Vue2.2.0新增API,这对选项需要一起使用，以允许一个祖先组件向其所有子孙后代注入一个依赖，不论组件层次有多深，并在起上下游关系成立的时间里始终生效。一言而蔽之：祖先组件中通过provider来提供变量，然后在子孙组件中通过inject来注入变量。

provide / inject API 主要解决了跨级组件间的通信问题，不过它的使用场景，主要是子组件获取上级组件的状态，跨级组件间建立了一种主动提供与依赖注入的关系。

Event Bus （$emit / $on ）（适用于 父子、隔代、兄弟组件通信）

这种方法通过一个空的Vue实例作为中央事件总线（事件中心），用它来触发事件和监听事件,巧妙而轻量地实现了任何组件间的通信，包括父子、兄弟、跨级。当我们的项目比较大时，可以选择更好的状态管理解决方案vuex。

Vuex（适用于 父子、隔代、兄弟组件通信）

11. 新增/删除vue对象属性视图不更新

受现代 JavaScript 的限制 (以及废弃 Object.observe)，Vue 不能检测到对象属性的添加或删除。由于 Vue 会在初始化实例时对属性执行 getter/setter 转化过程，所以属性必须在 data 对象上存在才能让 Vue 转换它，这样才能让它是响应的。

也就是说用原生js操控VUE中的对象，该对象不会更新到视图，解决方案如下：

添加/修改属性： Vue.set(object,key,val); 或者 this.$set(object,key,val);

删除属性： Vue.delete(object,key); 或者 this.$delete(object,key);

12. Vuex（状态管理模式，核心store，mutation 改变状态）

Vuex是通过全局注入store对象，来实现组件间的状态共享。在大型复杂的项目中（多级组件嵌套），需要实现一个组件更改某个数据，多个组件自动获取更改后的数据进行业务逻辑处理，这时候使用vuex比较合适。假如只是多个组件间传递数据，使用vuex未免有点大材小用，其实只用使用组件间常用的通信方法即可。

vuex有state,getter,mutation,action等关键属性，state主要是用于存放我们的原始数据结构，类似与vue的data,不过它是全局的，getter类似于计算属性computed,mutation主要用于触发修改state的行为，actions 也是一种触发动作，只不过与mutation的区别在于异步的操作我们只能在action中进行而不能在mutation中进行，目的是为了浏览器更好的跟踪state中数据的变化。

vuex是vue的状态管理器，存储的数据是响应式的。但是并不会保存起来，刷新之后就回到了初始的状态，具体做法应该在vuex里数据改变的时候吧数据拷贝一份保存到localStorage里面，刷新之后，如果localStorage里有保存的数据，取出来再替换store里的state。

属性：

State （定义了应用状态的数据结构，可以在这里设置默认的初始状态；数据源存放地，对应vue的data；响应式）
Getter （允许组件从 Store 中获取数据，mapGetters 辅助函数仅仅是将 store 中的 getter 映射到局部计算属性。）
Mutation （是唯一更改 store 中状态的方法，且必须是同步函数）
Action（用于提交 mutation，而不是直接变更状态，可以包含任意异步操作）
Module （允许将单一的 Store 拆分为多个 store 且同时保存在单一的状态树中）

12. Vue-router有哪几种导航钩子

1、全局前置钩子： router.beforeEach

2、全局解析守卫： router.beforeResolve

3、全局后置钩子： router.afterEach

4、路由独享的守卫： beforeEnter

5、组件内的守卫： beforeRouteEnter、beforeRouteUpdate (2.2 新增)、beforeRouteLeave beforeRouteEnter(to, from, next)

12. Vue-router的两种模式

对于 Vue 这类渐进式前端开发框架，为了构建 SPA（单页面应用），需要引入前端路由系统，这也就是 Vue-Router 存在的意义。前端路由的核心，就在于 —— 改变视图的同时不会向后端发出请求。

为了达到这一目的，浏览器当前提供了一下两种支持：
1：`hash - 即地址栏URL中的 # 符号` (此hash不是密码学里的散列运算)
比如这个URL：`http：//www.abc.com/#/hello`,hash的值为`#/hello`.它的特点在于：hash虽然出现在URL中，但不会被包括在HTTP请求中，对后端完全没有影响，因

此改变hash不会重新加载页面。

2：history - 利用了HTML5 History Interface中新增的`pushState()`和`replaceState（）`方法。（需要特定浏览器支持）

这两个方法应用于浏览器的历史记录栈，在当前已有的`back`、`forward`、`go`的基础上，它们提供了对历史记录进行修改的功能。只是当它们执行修改时，虽然改变了当前的URL，但浏览器不会即向后端发送请求。

因此可以说，hash模式和histoury模式都是属于浏览器自身的特性，Vue-Router只是利用了这两个特性（通过调用浏览器提供的接口）来实现前端路由

使用场景

一般场景下，hash 和 history 都可以，除非你更在意颜值，`#`符号夹杂在 URL 里看起来确实有些不太美丽。
如果不想要很丑的 hash，我们可以用路由的 history 模式，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。

结合自身例子，对于一般的 Vue + Vue-Router + Webpack + XXX 形式的 Web 开发场景，用`history 模式`即可，只需在后端（Apache 或 Nginx）进行简单的路由配置，同时搭配前端路由的 404 页面支持。

1.keep-alive

原理： keep-alive是一个抽象组件：它自身不会渲染一个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。

作用: 在做电商有关的项目中，当我们第一次进入列表页需要请求一下数据，当我从列表页进入详情页，详情页不缓存也需要请求下数据，然后返回列表页，这时候我们使用keep-alive来缓存组件，防止二次渲染，这样会大大的节省性能。

生命周期: 当引入keep-alive的时候，页面第一次进入，钩子的触发顺序created-> mounted-> activated，退出时触发deactivated。当再次进入（前进或者后退）时，只触发activated。

1.怎么做后台管理系统的页面权限控制

把路由相关信息配置在服务器，登录后获取到该角色对应的权限数据拼接好路由结构然后加入到路由配置中，据此显示该角色有权限操作的菜单，这种由后端返回数据的方式更易于维护。

1、什么是webpack，与grunt和gulp有啥不同

webpack是一个模块打包工具，在webpack里面一切皆模块
通过loader转换文件，通过plugin注入钩子，最后输出有多个模块组合成的文件

WebPack可以看做是模块打包机：它做的事情是，分析你的项目结构，找到Js模块以及其它的一些浏览器不能直接运行的拓展语言，并将其打包为合适的格式以供浏览器使用

Gulp/Grunt是一种能够优化前端的开发流程的工具，而WebPack是一种模块化的解决方案，不过Webpack的优点使得在很多场景下可以替代Gulp/Grunt类的工具

Grunt和Gulp的工作方式是：在一个配置文件中，指明对某些文件进行类似编译，组合，压缩等任务的具体步骤，工具之后可以自动替你完成这些任务

Webpack的工作方式是：把你的项目当做一个整体，通过一个给定的主文件（如：index.js），Webpack将从这个文件开始找到你的项目的所有依赖文件，使用 loaders处理它们，最后打包为一个（或多个）浏览器可识别的JavaScript文件

gulp和grunt需要开发者将整个前端构建过程拆分成多个Task，并合理控制所有Task的调用关系

webpack需要开发者找到入口，并需要清楚对于不同的资源应该使用什么Loader做何种解析和加工

2、webpack的优缺点

优点：

1. 专注于处理模块化的项目，能做到开箱即用，一步到位
2. 可通过plugin扩展，方便、灵活
3. 社区庞大活跃，经常引入新特性
4. 良好的开发体验

缺点：只能用于采用模块化开发的项目

vue图片懒加载实现步骤：

1.安装插件
npm install vue-lazyload --save-dev
2.main.js中引入，挂载
import Vuelazyload from 'vue-lazyload'
Vue.use(Vuelazyload, {
    error: require('../static/img/nonelive.png'),
    loading: require('../static/img/nonelive.png')
})
3.在渲染结构的地方把：src换成v-lazy即可

<img v-lazy="item.pictureUrl? item.pictureUrl:'../../../static/img/nonelive.png'" alt="图片未显示" onerror="this.src='../../../static/img/nonelive.png'">

vue项目搭建及全家桶的使用

一、全局安装node，webpack， vue-cli
二、构建工程文件：cmd中输入 vue init webpack projectName ，然后提示一些问题供填写和选择；
然后用npm install初始化项目，安装packaged.json文件中描述的依赖；
npm run dev 运行项目

在哪个生命周期内调用异步请求(created、beforeMount 、mounted 这三个钩子函数中data 已经创建，推荐created)

在什么阶段才能访问操作DOM（在mounted 阶段）

父组件监听子组件的生命周期（通过$emit触发父组件或父组件通过@hook 来监听）

组件中data为什么是函数（防止组件实例之间的data属性值不会互相影响）