Vue面试题100问

1.简述一下你对Vue的理解

> - Vue的作者是尤雨溪，一位华裔前Google工程师（Who）
> - Vue项目于2014年正式发布，至今已经有6年历史，而在2020年9月18日发布了3.0.0版本 One Piece。（When）
> - Vue是一个渐进式JavaScript 框架，它只负责视图层数据渲染功能，所以就需要很多第三方库来完善更多的扩展功能，以便完成动态构建用户界面的目标。（What）
> - Vue技术体系应用的场景非常的多，包括PC端SPA网站项目、后台管理系统、移动端Webapp的M站开发、微信小程序的实现以及App应用程序的处理等，应该说包含了多端多设备的不同应用面。（Where）
> - 这是因为Vue有其编码简洁、体积小、 运行效率高、遵循MVVM模式、轻松引入第三方插件与类库使用，能够快速实现应用程序的开发。（Why）
> - Vue在国内不同类型公司都广受欢迎，包括大型厂家，美团、饿了吗、阿里等，而中小型企业发展更追求速度与效率，所以对于Vue的需求也更为的明确。

2.声明式和命令式编程概念的理解

> - 声明式编程 Declarative Programing，主要关注“我想要什么”，而不关注具体该怎么实现，当你告诉“机器”你想要的是什么(what)，让机器想出如何去做(how)。声明式编程的计算在运行时构建完成。
> - 命令式编程 Imperative Programming，命令“机器”如何去做事情(how)，这样不管你想要的是什么(what)，它都会按照你的命令实现，计算机会严格遵循你的指令，而不理会最后的结果是不是你所想要的。命令式编程的计算在编译时构建完成。JavaScript并非完全是命令式编程，例如：数组的map方法，即为声明式编程。

3.Vue 有哪些基本特征

> - 声明式编程（没有DOM操作）
> - 响应式数据
> - 双向数据绑定

4.为什么组件中的data必须是函数形式?

> - Vue解析组件标签时，会创建一个新的组件实例对象
>
> 每个组件实例对象，都需要有自己的data数据对象
>
> 如果data配置是对象，就会导致同个组件的多个实例共享一个data对象
>
> 如果data是函数，组件的多个实例的data对象是各自的，是多份
> -
> -

5.Vue中v-for与v-if能否一起使用?

不能

> - 这需要考虑到v-for与v-if的优先级顺序问题，当它们处于同一节点，v-for的优先级比v-if更高，这意味着 v-if将分别重复运行于每个 v-for循环中

- 如果对于整体循环内容的控制，建议将条件判断编写至v-for循环的外部
- 如果需要对每个循环内容进行条件判断的话，建议编写computed属性计算进行对应内容的控制

6.vue中v-if与v-show的区别以及使用场景

区别

- 手段：v-if是通过控制dom节点的存在与否来控制元素的显隐；v-show是通过设置DOM元素的display样式，block为显示，none为隐藏；
- 编译过程：v-if切换有一个局部编译/卸载的过程，切换过程中合适地销毁和重建内部的事件监听和子组件；v-show只是简单的基于css切换；
- 编译条件：v-if也是惰性的，如果初始条件为假，则什么也不做；只有在条件第一次变为真时才开始局部编译； v-show是惰性的，在任何条件下都被编译，然后被缓存，而且DOM元素保留；
- 性能消耗：v-if有更高的切换消耗；v-show有更高的初始渲染消耗；

总结：
v-if判断是否加载，可以减轻服务器的压力，在需要时加载，但有更高的切换开销；v-show调整DOM元素的CSS的dispaly属性，可以使客户端操作更加流畅，但有更高的初始渲染开销。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。

7.v-on可以监听多个方法吗

可以，利用对象设置的方式设置多个监听事件

```
<input type="text" v-on="{ input:onInput, focus:onFocus,  blur:onBlur}">
```

8.v-on绑定的修饰符有哪些？v-model绑定的修饰符有哪些?

- v-on绑定的修饰符主要包括：事件的修饰符：stop、prevent、capture、self、once等，按键修饰符：enter、tab、delete、esc、space等，系统修饰键：ctrl、alt、shift等。
- v-model绑定的修饰符主要包括：lazy、trim、number

9.Vue中动态样式绑定的方式有哪些?

- 不管是class类样式还是style行内样式，要进行动态样式绑定都需要进行表达式的返回操作，而class绑定时表达式返回的结果可以是：字符串、对象、数组，并且数组中可以包含的内容包括字符串与对象类型。
- 而style绑定时表达式返回的结果是：对象、数组，数组中包含的是对象。

10.Vue能否通过下标的方式进行数组的响应式数据的修改？Vue能否通过路径的方式进行对象的响应式数据的修改？为什么?

不能

- 响应式数据需要在data中进行设置，而data中设置的响应式数据内容都会被转绑给vm实例对象，如果直接通过this.xxx设置的属性能够被挂载到Vue实例当中，但它们并不是响应式数据。
- 对于数组Vue内部对数组提供了一系列变异的方法操作，是对原来的Javascript数组操作进行了内部的重写，只不过Vue所提供的数组函数名称与Javascript所提供的名称保持一致而已，而这些数组的方法主要包括：push、pop、shift、unshift、splice、sort、reverse等

- Vue同样不能检测对象属性的添加或删除，但利用Vue.set(object, key, value) / Vue.delete(Object,key) 方法可以向嵌套对象添加响应式属性，你还可以使用 vm.$set / vm.$delete 实例方法，它只是全局 Vue.set / Vue.delete 的别名

11.如果后台接口返回的循环数据没有唯一值，在v-for循环中如何设置key?

key值在列表渲染的时候，能够提升列表渲染性能，为什么呢？首先得想想Vue的页面是如何渲染的，主要分为以下几步：

- 将页面结构的文档构建成一个vdom虚拟数
- 页面有新的交互，产生新的vdom数，然后与旧数进行比较，看哪里有变化了，做对应的修改（删除、移动、更新值）等操作（对比react、小程序）
- 最后再将vdom渲染成真实的页面结构

key值的作用就在第二步，当数据改变触发渲染层重新渲染的时候，会校正带有 key 的组件，框架会确保他们被重新排序，而不是重新创建，以确保使组件保持自身的状态，并且提高列表渲染时的效率。key值如果不指明，默认会按数组的索引来处理，因而会导致一些类似input等输入框组件的值出现混乱的问题。

- 不加key，在数组末尾追加元素，之前已渲染的元素不会重新渲染。但如果是在头部或者中间插入元素，整个list被删除重新渲染，且input组件的值还出现了混乱，值没有正常被更新
- 添加key，在数组末尾、中间、或者头部插入元素，其它已存在的元素都不会被重新渲染，值也能正常被更新

因而，在做list渲染时，如果list的顺序发生变化时，最好增加key，且不要简单的使用数组索引当做key，需要用唯一值当成key。

如果后台接口返回的循环数据没有唯一值，那么可以在客户端利用map循环对原数组内容进行遍历，然后返回经处理过带有唯一值的数组列表，那么在v-for循环的时候依旧设置的是唯一值数据。

12.为什么在-HTML-中监听事件?

- 扫一眼 HTML 模板便能轻松定位在 JavaScript 代码里对应的方法。
- 因为你无须在 JavaScript 里手动绑定事件，你的 ViewModel 代码可以是非常纯粹的逻辑，和 DOM 完全解耦，更易于测试。
- 当一个 ViewModel 被销毁时，所有的事件处理器都会自动被删除。你无须担心如何清理它们。

13.vue中methods、watch、computed之间的差别对比以及适用场景

methods,watch和computed都是以函数为基础的，但各自却都不同

- computed： 计算属性是基于其它的响应式依赖进行缓存的，只在相关响应式依赖发生改变时它们才会重新求值。就算在data中没有直接声明出要计算的变量，也可以直接在computed中写入。计算属性默认只有getter，可以在需要的时候自己设定setter。数据量大，并需要进行依赖计算的时候使用computed，因为它会进行缓存处理，提升执行的性能。computed应用的场景：一个数据受多个数据的影响。
- methods：首先compunted属性计算需要实现的目标利用methods方法都能够实现，但是compunted基于缓存处理，而methods每当触发重新渲染时，调用方法将总会再次执行函

数，从目标性能上来看，如果两者都可以实现目标的话一般建议使用compunted而不是methods。

- watch：watch和computed很相似，watch用于观察和监听页面上的vue实例，大部分情况下我们建议使用computed，但如果要在数据变化的同时进行异步操作或者是比较大的开销，那么watch为最佳选择。watch为一个对象，键是需要观察的表达式，值是对应回调函数。值也可以是方法名，或者包含选项的对象。如果在data中没有相应的属性的话，是不能watch的，这点和computed不一样。watch应用的场景：一个数据影响多个数据。watch监控除了编写watch对象，还可以利用vm.$watch进行API模式的监控操作。
- immediate：组件加载立即触发回调函数执行
- deep: 深度监听：为了发现**对象内部值**的变化，复杂类型的数据时使用，例如数组中的对象内容的改变

总结：在computed、methods、watch方面，一个是计算，一个是调用、一个是观察，在语义上是有区别的。计算是通过变量计算来得出数据，调用是方法的重复执行，而观察是观察一个特定的值。

14.computed属性计算函数能否传递参数?

可以

- compunted中定义属性计算函数，一般如果没有参数传递，可以直接return返回属性计算的结果
- 但如果需要传递参数，则计算函数中需要进行return fn(arg...)，而这个返回的函数中可以设置参数内容，最终属性计算的函数调用将会调用到return返回的这个函数内容。

```
{{ compuntedFn(arg) }}

computed: {
    compuntedFn: function () {
      return function (arg) {
        return "compuntedResult"
      };
    },
  },
```

15.Vue实例中最为重要的三大部分是什么?

不管是Vue也好或者其它的框架也罢，基本上面向对象以及组件化开发中实例对象都包含三个主要部分内容：

- 属性：设置属性，给实例对象内容进行属性的设值操作。
- 事件：触发事件，由对象提供事件，触发事件以后进行对应功能的处理。
- 方法：调用方法，方法的调用一般由第三方触发并调用对应的方法内容。

属性、事件是对象本身的操作，而方法一般交由第三方去操作对象本身内容，比如点击按钮去修改对象的属性操作。

16.组件在定义与使用的时候一般需要注意哪些细节?

- 组件中data的定义必须是一个函数：每个组件需要拥有自己的数据内容，并且不希望干扰到其它的组件内容，如果定义成Object对象的话，那么将会作用于所有的组件内容，在一个组件中进行data数据修改将会影响到其它组件。所以我们需要将它定义成函数的形式，函数返回的对象才是真正意义上数据的存储仓库(data)，这样设计的目的是为了让每个组件间的数据都是独立的，互不影响。
- 当组件的嵌套与Html5的规范产生冲突时我们可以使用is属性来解决冲突，比如：
- 组件上的所有事件都是vue自定义事件
- props特性属性与vue指令在组件中都不会继承，而非props特性属性则会进行继承处理

17.Vue中如何实现块状内容的输出?

块状内容在页面中将不会进行块状标签的输出

- Vue：template
- React：fragment，<>
- 小程序：block

18.简述一下组件的生命周期

在进行组件化项目开发的时候都会存在一个组件的生命周期概念，像Vue、React、小程序等等，无一例外，而通常情况组件的生命周期主要分成三个阶段，包括：创建、更新以及销毁阶段。

Vue的生命周期钩子函数主要包括：

1. beforeCreate(): 在实例初始化之后调用, data和methods都还没有初始化完成, 通过this不能访问
2. created(): 此时data和methods都已初始化完成, 可以通过this去操作, 可以在此发ajax请求
3. beforeMount(): 模板已经在内存中编译, 但还没有挂载到页面上, 不能通过ref找到对应的标签对象
4. mounted(): 页面已经初始显示, 可以通过ref找到对应的标签, 也可以选择此时发ajax请求
5. beforeUpdate(): 在数据更新之后, 界面更新前调用, 只能访问到原有的界面
6. updated(): 在界面更新之后调用, 此时可以访问最新的界面
7. beforeDestroy(): 实例销毁之前调用, 此时实例仍然可以正常工作
8. destroyed(): Vue 实例销毁后调用, 实例已经无法正常工作了
9. deactivated():组件失活, 但没有死亡
10. activated(): 组件激活, 被复用
11. errorCaptured(): 用于捕获子组件的错误,return false可以阻止错误向上冒泡(传递)

我们通常在created()/mounted()进行发送ajax请求，启动定时器等异步任务，而在beforeDestory()做收尾工作，如: 清除定时器操作。

不过需要注意的是mounted生命周期钩子中并不代表界面已经渲染成功，因为 mounted 不会保证所有的子组件也都一起被挂载。如果你希望等到整个视图都渲染完毕，可以在 mounted 内部使用 vm.$nextTick。

Vue的生命周期钩子函数又分为了：单个组件生命周期、父子组件的生命周期、带缓存的路由组件生命周期等不同的状态，在不同的状态下所拥有的生命周期内容是不相同的。

- 单个组件生命周期
  初始化：
  - beforeCreate
  - created
  - beforeMount
  - mounted

更新：

- beforeUpdate
- updated

销毁：

- beforeDestroy
- destroyed

- 父子组件的生命周期
  初始化:
  - beforeCreate
  - created
  - beforeMount
    - --child beforeCreate
    - --child created
    - --child beforeMount
    - --child mounted
  - mounted

  更新:
  - beforeUpdate
    - --child beforeUpdate
    - --child updated
  - updated

  销毁:
  - beforeDestroy
    - -- child beforeDestroy
    - -- child destroyed
  - destroyed

- 带缓存的路由组件生命周期
  初始化:
  - ...
  - mounted
    - --Child activated
  - activated

  路由离开
  - --Child deactivated
  - deactivated

  路由回来
  - --Child activated
  - activated

- 捕获子组件错误的勾子
  子组件执行抛出错误
  - errorCaptured

阶段	Vue	React	小程序应用	小程序页面
创建	beforeCreate	constructor()	onLaunch	onLoad
	created	static getDerivedStateFromProps()		onShow
	beforeMount	render()		onReady
	mounted	componentDidMount()		
更新	beforeUpdate	static getDerivedStateFromProps()	onShow	onShow
	updated	shouldComponentUpdate()	onHide	onHide
	deactivated	render()		
	activated	getSnapshotBeforeUpdate()		
		componentDidUpdate()		
销毁	beforeDestroy	componentWillUnmount()		onUnload
	destroyed			
捕获错误	errorCaptured	static getDerivedStateFromError()	onError	
		componentDidCatch()		

19.生命周期函数能否写成箭头函数？为什么？

不能。

- 在src引入Vue的操作模式中，如果生命周期函数写成箭头函数形式，函数体中的this指向将指向于Window，那将无法获取到Vue的实例对象，也就无法操作Vue的特性内容。而生命周期里面的函数（比如setTimeout）如果写成普通函数，那么将会改变this指向，所以需要写成箭头函数，以确保函数体内的this指向不被修改，指向于上层Vue实例对象内容。
- 在Vue脚手架项目的.vue组件文件中，生命周期钩子函数如果书写成箭头函数，this对象返回的是undefined，同样也无法进行Vue组件对象的获取。

20.哪一个生命周期钩子函数开始可以找到this对象内容？哪个钩子函数开始可以找到ref对象？哪个钩子函数开始可以找到DOM对象？

- 所有生命周期钩子函数都可以找到this对象内容，但却并不一定能够访问到想要的this对象属性，在beforeCreate实例初始化之后调用, data和methods都还没有初始化完成，通过this不能访问响应式数据与对应的方法内容，但因为store、router等对象内容是在入口文件中进行挂载的，那么在组件的beforeCreate钩子函数中是通过获取到对应的对象内容的，所以可以在beforeCreate钩子函数中进行类似route当前路由对象的参数内容获取操作。
- ref与Dom对象需要在mounted钩子函数中才能找到，而在mounted钩子函数中并不能确保页面都已经被渲染成功，所以还需要利用nextTick来进行DOM对象是否最终存在，而在destroyed钩子函数中因为组件的销毁，ref以及dom对象内容将不再能够获取。

21.组件间的关系类型有哪几种？有哪些方式可以实现组件之间的数据传递操作?

> 组件间的关系主要分成：父与子、子与父、祖与孙以及非父子之间的关系，正是因为存在组件之间的不同关系也就意味着它们之间会存在一定的数据传递操作，而组件之间的数据传递方式主要可以归纳为如下10多种实现：
>
> - 父子
>   - props：父向子、子向父
>   - vue自定义事件：子向父
>   - v-model：父子之间
>   - .sync：在父向子的基础上添加子向父
>   - $ref，$children与$parent
>   - $ref，$children：父向子
>   - $parent：子向父
>   - 插槽
>   - 默认插槽/具名插槽：父组件向子组件传递标签内容
>   - 作用域插件：子向父传递数据后，父组件根据接收到的数据来决定向子组件传递不同的标签内容
> - 祖孙
>   - provide与inject
>   - $attrs与$listeners
> - 非父子
>   - 全局事件总线
>   - Vuex

22.如何访问及修改根级组件的属性?

> - 不同于$parent、$children、$refs，$root获取的并不是VueComponent组件实例，而是Vue实例对象，所以我们不能够直接获取到this.$root的data数据内容以及调用this.$root的方法
> - 如果想要访问根级组件的数据可以利用this.$root.$children[0].xxx的方式进行获取， 利用this.$root.$children[0]的方式可以获取根级App组件，因为根级组件只有一个，所以取children的第1个元素。只要组件对象获取到了，那么它的数据获取、修改以及组件方法的调用都可以随心操作。

23.属性接收的约束以及注意事项有什么?

> - 子组件在进行props属性接收的时候可以利用数组及对象的方式进行处理，但数组方式只能进行简单的属性接收，而对像的方式则可以进行属性条件的约束操作。
> - 利用Object对象进行属性条件约束时需要注意default默认值的设置，default默认值在设置时必须从一个工厂函数获取，其原因与Vue组件的data设置是一个Function概念是一致的，因为需要确保数据的私有性问题。

24.Ref的作用与指向是什么?

> - 作用：ref 被用来给元素或子组件注册引用信息，引用信息将会注册在父组件的 $refs 对象上。
> - 指向：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。

25.Vue的动画方式有几种？需要注意的问题有哪些?

> Vue的动画方式主要分成两大类，一类是CSS动画，一类是JS动画
>
> - CSS动画中包含transition以及animation，但在Vue中只需要通过transition封装组件实现。
>   - CSS动画的类名主要包括：v-enter、v-enter-active、v-enter-to、v-leave、v-leave-active、v-leave-to
>   - transition只允许有一个元素内容，appear、type、duration、mode等属性可以进行动画操作的设置
>   - mode属性设置以后需要给动画元素设置唯一key值
>
>   in-out: 新元素先进行过渡，完成之后当前元素过渡离开。
>
>   out-in: 当前元素先进行过渡，完成之后新元素过渡进入。
>   - 一般情况下可以利用animate.css动画库内容进行CSS动画功能的实现
> - JS动画仍旧操作的是transition组件
>   - 设置的是属性钩子，内容包括before-enter、enter、after-enter、enter-cancelled、before-leave、leave、after-leave、leave-cancelled等
>   - 在 enter 和 leave 中必须使用 done 进行回调。否则，它们将被同步调用，过渡会立即完成
>   - 可以设置css属性为false，以免受css影响
>   - js动画同样可以利用js动画类库实现动画操作，比如Velocity.js
> - 列表动画可以利用transition-group进行实现

26.组件、过滤器、自定义指令的注册方式有哪几种?

> 组件、过滤器、自定义指令的注册方式都包含全局与局部注册两种
>
> - Vue.component、Vue.filter、Vue.directive进行的是全局注册，需要在入口文件中进行注册操作，所有组件可以进行使用。单词都是单数形式，需要单个内容的注册操作。
> - components、filters、directives则是在当前组件中进行局部注册，只能当前组件中进行对应功能的使用。单词形式都是复数形式，代表在组件中可以引入多个组件、过滤器与自定义指令内容。

27.Filter过滤器多个参数的传递以及多个过滤器的联合使用

> - 定义filter过滤器的函数第一个参数是需要过滤的对象内容，但假若在使用filter过滤器方法的时候想要传递参数，那么定义的filter过滤器的函数参数将从第一个参数内容开始获取。
> - 过滤器可以进行多个拼接使用，利用|管道符进行拼接，需要注意的是后面的过滤器操作的主体是前一个过滤器操作的结果值，特别需要注意前一个过滤结果的数据类型内容，以免引起类型操作错误。

28.Directive自定义指令的钩子函数以及函数参数是什么?

> 个指令定义对象可以提供如下几个钩子函数(均为可选)：bind、inserted、update、componentUpdated、unbind
>
> 对应钩子函数的参数主要包括：el、binding（属性：name、value、oldValue、expression、arg、modifiers）、vnode、oldVnode

29.封装自定义插件的操作步骤主要有哪些?

> - Vue.js 的插件应该暴露一个 install 方法，这个方法的第一个参数是 Vue 构造器，第二个参数是一个可选的选项对象，在这个方法中可以
>   - 添加全局方法或属性
>   - 添加全局资源
>   - 注入组件选项
>   - 添加实例方法
> - 使用插件的时候需要install安装以及use使用操作
> - 插件的类型主要包括：
>   - 对象插件: 调用插件对象install方法(传入Vue)来安装插件(执行定义新语法的代码)
>   - 函数插件: 直接将其作为install来调用(传入Vue)来安装插件(执行定义新语法的代码)

30.assets与static的共性与区别是什么?

> static和assets的区别，原理就在于**webpack是如何处理静态资源的**
>
> - **static**
> - static目录下的文件并不会被webpack处理，它们会直接复制到最终目录（dist/static）下。必须使用绝对路径引用这些文件，这是通过在config.js的build.assetsPublicPath和nuild.assetsSubDirectory连接确定的。
> - 任何放在static/的文件需要以绝对路径形式引用：/static/[name]。
>   如果更改assetsSubDirectory的值为assets，那么路径需更改为：/assets/[filename]。
> - assets
>   - 在vue组件中，所有模板和css都会被vue-html-loader和css-loader解析，并查找资源url。
>   例： 或者 background: url("./logo.png")
>   因为./logo.png是相对的资源路径，将会由webpack解析为模块依赖；
>   - 由于logo.png不是js，当被视作模块依赖时，需要使用url-loader和file-loader处理它，vue-cli已经配好了这些loader（webpack）因此可以使用相对/模块路径。
>   - 由于这些资源可能在构建过程中被内联、复制、重命名，所以它们基本是代码的一部分，即webpack处理的静态资源放在/src目录中，和其它资源文件放在一起。
>
> 总结：assets里面的资源会被webpack打包进代码，static里面的资源就直接引用了，一般在static里放一些类库的文件，assets放属于项目的资源文件。

31.Vue中如何实现表单数据的重置?

- Vue中的表单重置不像HTML的表单重置，只需要对form表单进行reset即可，因为Vue中的表单重置可能需要显示初始化的表单数据内容。
- 可以通过this.$data获取当前状态下的data，通过this.$options.data()获取该组件初始状态下的data。然后使用Object.assign(this.$data, this.$options.data())就可以将当前状态的data重置为初始状态。
- 如果需要需要重置某个data的节点属性，那么可以指明对应的节点内容 Object.assign(this.$data.formSelectObj, this.$options.data())

32.本地存储的方式有哪些？各自的优势与不足是什么?

- 实现本地存储的方式有很多，主要包括有localStorage、sessionStorage、indexedDb、webSql、Cookie，它们各有优势与不足
- Cookie是最为常用的一种本地存储方式，可以设置有效期限等条件限制，但有每个域名可以设置50个Cookie，每个不超过4K的文件大小，不便存储敏感信息
- localStorage是一种持久化存储，没有过期时间限制，但有尺寸大小限制，它有5M的存储空间
- 与localStorage相似的是sessionStorage，不过它是会话级存储，浏览器关闭则会清除相应的存储内容
- indexedDB是本地类似于mongodb一般的对象型存储方式，但对浏览器版本要求比较高，可以进行类似mongodb一般的数据增、删、改、查操作，属于无限量空间存储。
- webSql也是本地数据库存储模式，同样对浏览器版本要求比较高，但它则与mysql关系型数据库操作非常相似，也可以进行数据的sql语法操作，进行增、删、改、查，非常的方便，也属于无限空间存储模式。

33.什么是插槽，有哪些类型?

插槽是一种组件间html传递的策略，实现父组件向子组件传递标签内容。插槽的类型主要包括：普通插槽、具名插槽以及作用域插槽

对于作用域插槽是在父组件需要向子组件传递标签结构内容，但决定父组件传递怎样标签结构的数据是在子组件中。在Vue2.6版本以后，新版本插槽的语法比起老版本区别略有区别，主要可以利用#slotName={property}的方式进行简化缩写。

```
slot="插槽的名字" + slot-scope="{要收集的数据-1,要收集的数据-2}"
                =
v-slot:插槽名字="{要收集的数据-1,要收集的数据-2}"
                =
#插槽名字="{要收集的数据-1,要收集的数据-2}"
```

34.组件的类型及特点有哪些?

如果想要对组件进行类型划分，从实现的功能以及所具备的特点来划分，大致可以归纳为：动态组件、缓存组件、异步组件、函数式组件 + JSX、递归组件等

- 动态组件：通过动态确定要显示的组件, is指定要显示组件的组件名

  ```
  <component :is="currentComp" />
  ```

  问题: 当从A组件切换到B组件时, A组件就会销毁

- 缓存组件：即可以利用component，也可以使用router-view

    - 使用缓存动态组件, 可以通过include或exclude指定只缓存特定组件

```
<keep-alive :exclude="['Home']">
    <component :is="currentComp"/>
</keep-alive>
```

    - 使用也可以缓存路由组件

```
<keep-alive include="Life1">
    <router-view></router-view>
</keep-alive>
```

    路由组件对象什么时候创建?

    - 默认: 每次跳转/访问对应的路由路径时
    - 有缓存: 第一次访问时

    路由组件对象什么时候死亡?

    - 默认: 离开时
    - 有缓存: 离开时不死亡, 只有当destroy/父组件死亡/刷新页面

- 异步组件
    - 好处：能更快的看到当前需要展现的组件界面(异步组件的代码开始没有加载)
    - 无论是**路由组件**还是**非路由组件**都可以实现异步组件效果
        - 拆分单独打包
        - 需要时才请求加载组件对应的打包文件
    - 配置组件: component: () => import(modulePath)
        - import(modulePath): 被引入的模块会被单独打包(code split) --ES8的新语法
        - () => import(modulePath): 函数开始不调用, 当第一次需要显示对应的界面时才会执行, 请求加载打包文件
    - 细节
        - import()返回promise, promise成功的结果就是整体模块对象
        - 本质上: 可以利用import()实现对任意模块的懒加载
- 函数式组件 + JSX，与Vue的Component概念有明显不同，更雷同于React中的函数式组件
    - 只能针对无状态(data)的组件
    - 不用创建实例对象, 运行更快
    - 可以没有根标签

```
export default {
    functional: true, // 当前是函数组件
    render (createElement, context) {
        return 要显示界面的虚拟DOM
    }
}
```

- 递归组件，组件内部有自己的子组件标签，简单的说就是组件自己调用自身
    - 应用场景: 用于显示树状态结构的界面
    - 注意: 递归组件必须有name

35.简述一下对Vue-Router的理解?

> 在以往的项目开发中，包括使用不同技术栈的项目内容，例如nodejs的express、Vue项目、React项目、小程序项目等都涉及到了路由的概念与操作。虽然这些项目归属于不同的技术体系，但路由的核心概念都是一致的，我也做了相应的归纳，总结出5个词进行了概括：静态路由表、分配地址、统一入口、寻址渲染，过滤判断。当然，对于不同的技术体系，路由的表现与配置方式会有所差异与不同。
>
> 路由的具体操作又表现在：操作模式、跳转方式、参数的传递、嵌套处理、守卫管理、懒加载及动态路由等方面。

功能	React-router	Vue-Router
静态路由表及分配地址	HashRouter/BrowserRouter as Router Route/Switch	设置数组，path和component对象内容的确认
首页的渲染	Route的path为/，但是需要加上exact	router-view
链接形式1	a链接，但是要加#	a链接，但是要加#
链接形式2	Link/NavLink链接，to，不需要加#	router-link,to，不需要加#
高亮显示	NavLink下的activeClassName/activeStyle	active-class
路由嵌套	是通过不同的组件中设置Route来进行路由的嵌套，嵌套操作被拆分了	children嵌套
路由嵌套的显示	Route既是路由，也是占位渲染显示	router-view
参数传递	设参在路由 :xxx传参在地址 Link/NavLink to的设置 接参在组件 this.props.match.params.xxx	设参在路由 :xxx传参在地址 router-link的to的设置 接参在组件 this.$route.params.xxx/watch
首页的高亮显示	exact	exact
多层嵌套	是通过不同的组件中设置Route来进行路由的嵌套，嵌套操作被拆分了	children的多层嵌套+router-view
程序式导航	this.props.history.push(pathUrl)	this.$router.push(location)
Miss与NoMatch	NoMatch不需要写path，直接写NoMatch	redirect与*号通配符

36.路由守卫的类型有哪些，一共有几个路由守卫，不同类型包含哪些路由守卫的钩子函数

> 导航守卫是什么?
>
> - 导航守卫是vue-router提供的下面2个方面的功能
>   - 监视路由跳转 -->回调函数
>   - 控制路由跳转 --> 放行/不放行/强制跳转到指定位置 next()
> - 应用
>   - 在跳转到界面前, 进行用户权限检查限制(如是否已登陆/是否有访问路由权限)
>   - 在跳转到登陆界面前, 判断用户没有登陆才显示
>
> 导航守卫分类主要包括：
>
> - 全局守卫：beforeEach、beforeResolve、afterEach

- 路由独享的守卫：beforeEnter
- 组件内的守卫：beforeRouteEnter、beforeRouteUpdate、beforeRouteLeave

当点击切换路由时：beforeRouterLeave-->beforeEach-->beforeEnter-->beforeRouteEnter-->beforeResolve-->afterEach-->beforeCreate-->created-->beforeMount-->mounted-->beforeRouteEnter的next的回调

当路由更新时：beforeRouteUpdate

37.最为常用的路由守卫类型是哪个，主要应用在什么功能操作?

最为常用的路由守卫应当是全局守卫中的beforeEach，因为在用户权限认证操作过程中都会需要该守卫操作的处理，而用户权限又是每个项目中不可缺少的一部分。

- 关键技术: 全局前置守卫beforEach + 动态添加路由
- 判断是否有token
    - 没有token, 判断请求是否是白名单路由:
        - 是: 直接放行
        - 不是: 强制跳转到login页面
    - 有token, 判断请求的是否是login页面
        - 是: 强制跳转到根路由
        - 不是, 判断是否已经登陆?
            - 已登陆: 放行
            - 没有登陆:
                - 请求获取用户相关信息数据: name/avatar/路由的权限数据和按钮的权限数据
                - 请求成功了:
                    - 将用户相关数据保存到vuex中
                    - 根据路由权限数据动态生成权限路由的数组
                    - 将所有的权限路由和匹配任意路由的路由动态添加到router中
                - 请求失败了:
                    - 删除cookie中和token
                    - 删除vuex中用户相关信息
                    - 强制跳转到登陆页面

38.router与route的差异与区别是什么?

- 我们可以根据这两个单词的长度来进行判断，一个是router，这个单词长一些，代表的是整个路由对象，里面是整个静态路由表的配置信息。如果需要实现路由的跳转，则需要从整个静态路由表对象中进行匹配，所以使用的是router。
- 而route这个单词短一些，代表的是当前的路由页面内容，如果需要获取当前路由对象的参数信息，则可以对route内容进行处理。

39.Vue的双向数据绑定原理是什么?

答：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 这样的话，给这个对象的某个值赋值，就会触发setter，那么就能监听到了数据变化

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。

40.对keep-alive 的了解

keep-alive是 Vue 内置的一个组件，可以使被包含的组件保留状态，或避免重新渲染。它有两个属性: include(包含的组件缓存) 与 exclude(排除的组件不缓存，优先级大于include) 。

使用方法

```
<keep-alive include='include_components' exclude='exclude_components'>
<component>
 <!-- 该组件是否缓存取决于include和exclude属性 -->
</component>
</keep-alive>
```

参数解释
include - 字符串或正则表达式，只有名称匹配的组件会被缓存
exclude - 字符串或正则表达式，任何名称匹配的组件都不会被缓存
include 和 exclude 的属性允许组件有条件地缓存。二者都可以用“，”分隔字符串、正则表达式、数组。当使用正则或者是数组时，要记得使用v-bind 。

使用示例

```
<!-- 逗号分隔字符串，只有组件a与b被缓存。 -->
<keep-alive include="a,b">
<component></component>
</keep-alive>

<!-- 正则表达式 (需要使用 v-bind，符合匹配规则的都会被缓存) -->
<keep-alive :include="/a|b/">
<component></component>
</keep-alive>

<!-- Array (需要使用 v-bind，被包含的都会被缓存) -->
<keep-alive :include="['a', 'b']">
<component></component>
</keep-alive>
```

41.路由切换以后需要进行页面中某一锚点的定位需要如何实现?

> 需要利用router的scrollBehavior来进行指定位置的跳转操作

```
scrollBehavior: function (to) {
 if (to.hash) {
     return {
         selector: to.hash
     }
 }
},
```

42.你如何理解vuex?

> **vuex的6大属性:**

内容	作用	映射	位置	调用	其它
state	设置状态	mapState	computed		
getters	获取内容	mapGetters	computed		计算数据后返回
mutations	修改数据	mapMutations	methods	commit	可以异步但不建议，不利调试
actions	异步操作	mapActions	methods	dispatch	
modules	模块拆分				namespaced
plugins	插件辅助				

43.vuex中的namespaces是什么？它的主要作用是什么?

> 在Vuex模块中开启 namespaced以后，确定该模块为带命名空间的模块。当模块被注册后，它的所有 state、getter、action 及 mutation 都会自动根据模块注册的路径调整命名。

44.如何解决vuex数据丢失问题?

> 刷新页面Vuex的state会丢失，可以利用vuex-persist、vuex-persistedstate状态持久化插件将state数据存储于本地存储对象当中，比如localStorage。

45.vuex和redux的区别?

> - vuex:
>   - 我们直接在mutation中直接更新状态数据
>   - 直接可以在action中执行异步操作
> - redux:
>   - 只能通过reducer返回一个新的状态数据, 由redux内部自己更新
>   - 本身不支持异步, 必须引入react-redux之类的插件才支持异步

46.如果不使用分页处理，在一个页面中想显示10万条数据，如何实现高效的DOM渲染处理?

> - 利用虚拟滚动的方式，控制在指定区域内容只渲染可显示范围的数据内容，不增加DOM节点
> - vue-virtual-scroller/vue-virtual-scroll-list第三方插件的应用

47.前端开发人员在不借助后端人员协助的情况下，如果换作是你，如何最快速的实现跨域问题的解决？跨域问题的解决一般有哪些方式，如何实现?

> - 直接安装Chorme插件，CORS unblock开启插件即可
> - 利用vue.config.js配置devServer的proxy反向代理
> - 通过建立本地nginx服务器，配置代理地址

48.一般情况，你会对程序的错误分成几大类型？你是如何进行程序的调试的处理的?

> - 语法错误：利用编辑器进行提示，包括eslint等辅助工具的配置使用
> - 运行错误：利用报错提示查看出错的行列，分析错误情况进行排错
> - 逻辑错误：梳理业务流程，利用debug调试工具，包括断点测试实现纠错

49.你如何快速了解、学习与掌握脚手架项目或者其他公司新接收的项目?

> 项目的发解、学习与掌握也是有一定的顺序与方法的，主要的流程包括如下几个步骤：
>
> - 有Readme说明文档一定先看Readme说明文档，最为主要解决的是项目运行环境的搭建与启动
> - 项目目录结构的分析
> - 项目文件结构的分析
> - 项目代码结构的分析：需要找到入口文件以及主组件，从入口文件着手，主组件触发，从上到下，剥洋葱一般的进行代码层次结构的剥离与分析

50.对于Vue项目，你所常用的性能优化方式主要包含哪些方面?

> 项目性能优化的方面包含很多，针对Vue项目的优化可以介绍几种类型的内容，主要包括：Vue 代码层面的优化、webpack 配置层面的优化、基础的 Web 技术层面的优化、用户体验优化等
>
> 代码层面的优化：
>
> - v-if 和 v-show
> - computed 和 watch
> - 不要将所有的数据放在data中，固定数据或者定时器等可设置于实例对象当中
> - keep-alive缓存组件
> - v-for 遍历key设置，且避免同时使用 v-if
> - 长列表利用Object.freeze冻结数据
> - 监听对象的销毁
> - 图片资源懒加载
> - 路由懒加载，异步组件
> - 第三方插件的按需引入
> - 无限列表利用虚拟滚动列表实现
> - 服务端渲染或预渲染
> - 函数式组件应用

> - 高频触发使用防抖、节流
> - 事件委托
> - 图片编码优化，尽量使用svg和字体图标
> - 避免重定向以及404页面
> - 动态注册组件
> - 使用编程式导航代替声明式导航
> - 前端表单验证，减少请求处理
>
> Webpack 层面的优化
>
> - 文件压缩，包括图片、css、js、html等
> - 减少 ES6 转为 ES5 的冗余代码
> - 提取公共代码
> - 模板预编译
> - 模块文件的提取，包括css、第三方js库等
> - 优化 SourceMap
> - 构建结果输出分析
>
> 基础的 Web 技术层面的优化
>
> - 开启 gzip 压缩
> - 浏览器缓存
> - CDN 的使用
> - 使用 Chrome Performance 查找性能瓶颈

https://www.jianshu.com/p/ef44aaa41fe8

51.路由跳转指定params参数时可不可以用path和params配置的组合?

> 不可以，只能用name和params配置的组合，query配置可以与path或name进行组合使用

52.如何指定params参数可传可不传?

> path: '/search/:keyword?'，利用?号设定

53.如果指定name与params配置, 但params中数据是一个"", 无法跳转

> 不指定params或者指定params参数值为undefined

54.路由组件能不能传递props数据?

> 可以：可以将query或params参数映射成props传递给路由组件对象
>
> ```
> //在routes中配置
> props: route=>({keyword1:route.params.keyword, keyword2: route.query.keyword
> })
> ```

55.编程式路由跳转到当前路由(参数不变), 会抛出NavigationDuplicated的警告错误

面试问题:：在做项目时有没有遇到比较难的问题?(可做回答)

回答步骤:

1. **我的问题**: 我在上一个项目时没有问题，后面再做一个新的项目时就有了问题
2. **原因分析**: vue-router3.1.0之后，引入了push()的promise的语法，如果没有通过参数指定回调函数就返回一个promise来指定成功/失败的回调，且内部会判断如果要跳转的路径和参数都没有变化，会抛出一个失败的promise
3. **解决办法**：解决1: 在跳转时指定成功或失败的回调函数, 通过catch处理错误

   解决2: 修正Vue原型上的push和replace方法 (优秀)

```
// 缓存原型上的push方法
const originPush = VueRouter.prototype.push
VueRouter.prototype.push = function (location, onComplete, onAbort) {
  console.log('push()', location, onComplete, onAbort)
  // this是路由器对象 $router
  // 如果调用push, 传递了成功或者失败的回调函数
  if (onComplete || onAbort) {
    // 让原来的push方法进行处理
    originPush.call(this, location, onComplete, onAbort) // 不用返回, 因为执行的结果返回是undfined
  } else { // 如果调用push, 没传递了成功或者失败的回调函数, 可能会抛出失败的promise, 需要catch一下
    return originPush.call(this, location).catch(() => {
      console.log('catch error')
    })   // 必须返回产生的promise对象
  }
}
```

56.是否有对axios进行二次封装？主要的封装功能包括哪些?

1. 配置通用的基础路径和超时:

   axios.create({baseURL, timeout})
2. 显示请求进度条

   显示: 准备发请求前显示, 在请求拦截器中执行NProgress.start()

   隐藏: 请求结束隐藏, 在响应拦截器成功/失败回调中NProgress.done()
3. 携带token数据

   在请求拦截器中, 将token添加到请求头中
4. 成功返回的数据不再是response, 而直接是响应体数据response.data

   响应拦截器成功的回调中: return response.data
5. 统一处理请求错误, 具体请求也可以选择处理或不处理

   在响应拦截器失败的回调中: alert提示错误信息, return Promise.reject(error)

56.为什么要使用编程式导航代替声明式导航，典型应用场景是什么?

> 在重复使用声明式导航时需用编程式导航替换，以便提升性能。在电商项目商品分类跳转时可以进行编程式导航的应用。

57.如何配置使用swiper?

> - 必须在列表显示显示之后创建 , 在mounted()中创建
> - 异步动态获取数据并且在mounted中创建Swiper对象也会导致没有轮播效果，需要nextTick确定DOM已经渲染完毕

58.如何解决多个swiper效果冲突的问题

> - 问题: 针对某个swiper界面创建一个swiper对象, 它会影响了其它界面的swiper界面
> - 原因: new Swiper ('.swiper-container'), 类名选择器匹配了页面中所有的swiper界面, 都产生了效果
> - 解决: 使用ref技术: 通过ref标识swiper的根div, new Swiper (this.$refs.swiper)

59.什么情况下需要深度作用选择器修改第三方UI组件的内部样式，如何实现

> 场景：当我们需要覆盖element-ui等UI框架中组件的样式时可以通过深度作用选择器
>
> **style为css时的写法如下**
>
> ```
> .a >>> .b {...}
> ```
>
> **style使用css的预处理器(less, sass, scss)的写法如下**
>
> ```
> /deep/ .a {...}
> ```
>
> ```
> /* /deep/在某些时候会报错,::v-deep更保险并且编译速度更快 */
> ::v-deep .a{...}
> ```

60.利用深拷贝解决修改不能取消的问题

> 在对某数据进行修改时考虑还需要进行“确认”、“取消”操作，那么在取消时就需要返回保留的数据内容，那么如何将原有数据保留一份则是关键性问题。
>
> - 显然修改值不能直接进行原值的赋值操作，因为这样无法取消回退
> - 如果采用浅拷贝，那么浅拷贝只复制指向某个对象的指针，而不复制对象本身，新旧对象还是共享同一块内存，那么数据还是会出现问题
> - 需要采用深拷贝的形式进行数据的复制
>   - JSON.parse(JSON.stringify(obj))实现深拷贝其实会存在很多的问题
>   - 如果obj里面有时间对象，则JSON.stringify后再JSON.parse的结果，时间将只是字符串的形式，而不是时间对象
>   - 如果obj里有RegExp、Error对象，则序列化的结果将只得到空对象
>   - 如果obj里有函数，undefined，则序列化的结果会把函数或 undefined丢失
>   - 如果obj里有NaN、Infinity和-Infinity，则序列化的结果会变成null

        - JSON.stringify()只能序列化对象的可枚举的自有属性，例如 如果obj中的对象是有构造函数生成的， 则使用JSON.parse(JSON.stringify(obj))深拷贝后，会丢弃对象的constructor
        - 如果对象中存在循环引用的情况也无法正确实现深拷贝
    - 使用lodash的cloneDeep进行数据的深拷贝

61.vue变量名如果以_、$开头的属性会发生什么问题？怎么访问到它们的值?

- 以_或者$开头的属性不会被vue实例代理，因为它们可能会和vue内置属性、api方法冲突
- 可以使用vm.$data._property的方式访问这些属性。

62.Vue组件中写name选项有什么作用

- 使用keep-alive时，可以搭配组件name进行缓存过滤
- DOM做递归组件时需要调用自身name。
- vue-devtools调式工具里显示的组件名称是有vue中name决定的

63.Vue渲染模板时怎么保留模板中的HTML注释呢?

```
// 模板标签添加 comments 属性
<template comments>
  ...
</template>
```

64.如何更优雅的进行监听对象的清除处理?

可以通过$once这个事件侦听器在定义完定时器之后的位置来清除定时器

```
mounted(){
const timer = setInterval(()=>{
 console.log(1)
},1000)
this.$once('hook:beforeDestroy',()=>{ // 监听beforeDestroy这个钩子函数
clearInterval(timer)
})
}
```

65.请说一下computed中的getter和setter

computed 中可以分成 getter（读取） 和 setter（设值），一般情况下是没有 setter 的，computed 预设只有 getter，也就是只能读取，不能改变设值。

- 默认只有 getter的写法

```
<div id="demo">{{ fullName }}</div>
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar'
  },
```

```
  computed: {
    fullName: function () {
      return this.firstName + ' ' + this.lastName
    }
  }
})
//其实fullName的完整写法应该是如下：
fullName: {
 get(){
   return this.firstName + ' ' + this.lastName
 }
}
```

注意：不是说我们更改了getter里使用的变量，就会触发computed的更新，前提是computed里的值必须要在模板里使用才行。如果将{{fullName}}去掉，get（）方法是不会触发的。

- setter的写法，可以设值

```
<template>
   <div id="demo">
       <p> {{ fullName }} </p>
       <input type="text" v-model="fullName">
       <input type="text" v-model="firstName">
       <input type="text" v-model="lastName">
   </div>
</template>

var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'zhang',
    lastName: 'san'
  },
  computed: {
    fullName: {
      //getter 方法
     get(){
       console.log('computed getter...')
        return this.firstName + ' ' + this.lastName
       },
   //setter 方法
    set(newValue){
      console.log('computed setter...')
      var names = newValue.split(' ')
      this.firstName = names[0]
      this.lastName = names[names.length - 1]
      return this.firstName + ' ' + this.lastName
     }
    }
  }
})
```

在这里，我们修改fullName的值，就会触发setter，同时也会触发getter。

> 注意：并不是触发了setter也就会触发getter，他们两个是相互独立的。我们这里修改了fullName会触发getter是因为setter函数里有改变firstName 和 lastName 值的代码，这两个值改变了，fullName依赖于这两个值，所以便会自动改变。

66.如果elementUI，iview等UI框架的Table表格组件功能达不到业务复杂度需求，比如需要实现树形结构的深度查询、分组汇总以及合计、行列单元格的可编辑操作等，会如何操作?

> elementUI，iview是比较符合大众业务需求的UI框架，而表格组件在业务需求的使用中极为的频繁，elementUI，iview对于Table表格组件的功能并没有做极大的强化，所以可以考虑选择更为专业的vxe-table插件，https://xuliangzhan_admin.gitee.io/vxe-table/#/table/start/install

67.利用Vue技术体系是否能够进行小程序项目的开发?

> 可以
>
> - wepay、mpx、mpvue（停更）、uniApp都是类Vue或者是Vue体系的技术框架，利用它们可以进行小程序项目的开发。

68.利用Vue技术体系是否能够进行App项目的开发?

> 可以
>
> - 阿里的weex，dclould的uniApp，还有NativeScript的Vue体系都可以进行移动端App项目的开发。

69.如何实现路由参数的响应式处理?

> 可以利用监控或者组件的路由守卫功能实现
>
> 监控模式：
>
> ```
> const User = {
> template: ' User {{ $route.params.name }} ',
> watch: {
>  '$route' (to, from) {
>    // react to route changes...
>  }
> }
> }
> ```
>
> 路由守卫模式：
>
> ```
> const User = {
> template: ' User {{ $route.params.name }} ',
> beforeRouteUpdate (to, from, next) {
>  // react to route changes and then call next()
> }
> }
> ```

70.在同一组件中能否既定义全局样式，又定义组件局部样式?

> 可以，定义两个style，一个没有scoped，一个有scoped

71.Vue中多国语言支持需要利用什么方式实现?

> 可以利用Vue-i18n插件实现：http://kazupon.github.io/vue-i18n/

72.能否将vue-router 当前的 $route 同步为 vuex 状态的一部分?

> 可以，利用vuex-router-sync可以将vuex与router对象进行同步，将当前的$route同步为vuex状态的一部分，我们甚至可以利用修改vuex的路由状态来进行路由地址的跳转与参数传递
>
> ```
> commit('route/ROUTE_CHANGED',{to: {path: '/b'}})
> ```

73.如何对lodash库实现按需引入

```
import _ from 'lodash'  // 引入整体lodash  ==> 打包了没用的工具函数，打包文件变大
import throttle from 'lodash/throttle'  // 只引入我需要的工具函数   打包文件减少
1.4M
```

74.如何实现跨域配置

> 在vue.config.js中对devServer进行属性配置
>
> ```
> proxy: {
>  '/dev-api': { // 匹配所有以 '/dev-api'开头的请求路径
>      target: 'http://182.92.128.115', // 代理目标的基础路径
>      changeOrigin: true, // 支持跨域
>      pathRewrite: { // 重写路径: 去掉路径中开头的'/dev-api'
>          '^/dev-api': ''
>      }
>  },
> }
> ```

75.理解Vue.use

> - 自定义Vue插件需要向外暴露对象或者是函数
> - 如果向外暴露对象的话，对象中必须有install方法
> - 如果向外暴露的是函数的话，那么该函数本身就是install方法
> - 当Vue.use()的时候，会自动调用install方法，并且将Vue对象作为实参传入到install方法中

75.history模式刷新404问题解决方法

> - 通过配置webpack来解决：在devServer中加 historyApiFallback: true
> - index.html 需要将 href="./bootstrap.css" 等内容改成 href="/bootstrap.css"
> - webpack.config.js 需要output上增加 publicPath: '/'

76.v-model都做了哪些事情

> - 将指定变量的数据赋值给input的value
> - 给当前的表单相自动绑定一个input事件，监听View层表单项数据发生改变获取最新value的同时更新Model的数据

77.路由设置中的meta属性通常设置什么内容?

> 一般我们会设置title标题以及isAuth是否授权操作等自定义的路由属性内容

78.如何动态修改Vue单页面应用的标题

> - 通过设置路由的meta属性，添加路由title标题
> - 利用路由全局守卫可以进行拦截并修改标题设置
>
> ```
> router.beforeEach = ((to,from,next)) => {
>     window.document.title = to.meta.title;
>     next();
> });
> ```

79.什么是Props传递数据防脏

> 所有的 props 都使得其父子组件形成一个单向下行绑定，父级 props 的更新会流动到子组件中，但反过来不行。这种设计办法是为了防止子组件意外改变父组件的状态，从而导致你的应用的数据流向难以理解。另外，如果该数据还被其他子组件使用，也将受影响，产生洪水式灾难。因此不应该在子组件中设计修改 props 数据的操作。

80.如何实现数据防脏操作?

> 利用update:myPropName的方式进行数据防脏处理
>
> - 子组件赋值操作
>
> ```
> this.$emit('update:title', newTitle)
> ```
>
> - 父组件事件绑定处理
>
> ```
> <text-document
>   v-bind:title="doc.title"
>   v-on:update:title="doc.title = $event"
> ></text-document>
> ```
>
> - 缩写模式，利用sync
>
> ```
> <text-document v-bind:title.sync="doc.title"></text-document>
> ```

81.你所熟悉与应用的VueUI框架有哪些?

> - PC端主要有iview、elementUI、Ant-Design-Vue
> - 移动端：MintUI、Vant、Vux

82.如果实现css的模块化?

> 使用 CSS Modules实现样式的模块化

```
<!-- 使用 CSS Modules -->
<style module>
.button {
border: none;
border-radius: 2px;
}

.buttonClose {
background-color: red;
}
</style>
```

83.如何实现事件委托操作?

> 利用事件冒泡将事件操作委托于父元素的事件对象内容

```
<!-- 不将事件绑定于button,而是绑定在table上 -->
<table @click="edit">
  <tr v-for="item in list" :key="item.id">
      <td>{{ item.name }}</td>
      <td>
          <button :data-id="item.id" title="edit">编辑</button>
      </td>
  </tr>
</table>
```

```
// 通过获取绑定的标识参数来确认事件目标对象
edit(event) {
  if (event.target.title == "edit") {
      //如果点击到了edit
      let id = event.target.dataset.id;
  }
},
```

84.如何实现模块自动化加载操作?

> require.context()函数获取一个特定的上下文，主要用来实现自动化导入模块，在前端工程中，如果遇到从一个文件夹引入很多模块的情况,可以使用这个api,它会遍历文件夹中的指定文件,然后自动导入,使得不需要每次显式的调用import导入模块

```
/*
动态加载vuex中所有的modules模块
不再需要通过import手动一个一个引入
*/
const context = require.context('./modules', false, /\.js$/)
const modules = context.keys().reduce((modules, modulePath) => {
// './app.js' => 'app'
const moduleName = modulePath.replace(/^\.\/(.*)\.\w+$/, '$1')
modules[moduleName] = context(modulePath).default
return modules
}, {})
```

85.addRoutes的作用与应用场景

> addRoutes可以实现动态路由的添加操作
>
> 应用场景：根据用户权限判断展示不同路由菜单项
>
> - 前台公共路由的获取
> - 后台通过用户权限判断返回具体用户拥有的路由
> - 前台利用addRoutes将公共路由与后台返回的动态路由进行组合成新的用户功能路由

```
router.addRoutes([...asyncRoutes, lastRoute]);
```

86.不同路由同一组件的重用

> 如果设置不同的路由，但指向的是同一组件，那么因为组件性能考虑及缓存策略，组件并不会刷新，那么如何实现组件的刷新重用呢？

```
<template>
    <router-view :key="$route.fullPath"></router-view>
</template>
```

87.如何实现组件属性的验证处理?

> 可以利用 validator 进行自定义验证方法的定义处理

```
alertType: {
validator: value => ['signup', 'login', 'logout'].includes(value)
}
```

88.如何实现组件的刷新?

> 一般情况组件的state状态或者是props发生改变时就会进行刷新渲染，但有时需要进行用户控制，那么有哪些方法呢：
>
> - 最差的方式：刷新页面
> - 常见的方法：v-if控制组件的显示
> - 利用forceUpdate强制更新，将触发updated钩子函数

```
<template>
  <div id="app">
    <h1>{{Math.random()}}</h1>
    <button @click="update">Force Update</button>
```

```
  </div>
</template>

<script>
export default {
  methods:{
    update(){
      this.$forceUpdate();
    }
  }
};
</script>
```

- 利用mount进行重新挂载，将触发beforeMount与updated钩子函数

```
<div id="app">
    <h1>{{Math.random()}}</h1>
    <button @click="update">Force Update</button>
  </div>
</template>

<script>
export default {
  methods:{
    update(){
      this.$mount();
    }
  }
};
</script>
```

89.重载刷新页面

- this.$router.go(0) 相当于F5刷新，这种方法虽然代码很少，只有一行，但是体验很差。 页面会一瞬间的白屏，体验不是很好
- location.reload() 这种也是一样，画面一闪，体验不是很好，相当于页面刷新
- 推荐解决方法：用provide / inject 组合

在App.vue,声明reload方法，控制router-view的显示或隐藏，从而控制页面的再次加载。

```
<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>
```

在需要用到刷新的页面。在页面注入App.vue组件提供（provide）的 reload 依赖，在逻辑完成之后（删除或添加...），直接this.reload()调用，即可刷新当前页面。

注入reload方法

```
export default {
   inject: ['reload'],
}
```

在需要重载的地方直接调用

```
this.reload()
```

90.表单提交时如何阻止页面刷新?

表单提交时为了防止页面刷新需要进行默认事件的阻止处理，可以利用submit.prevent进行实现。

91.是否有集成过在线编辑器？使用的是哪些?

百度ueditor的Vue版本，vue-ueditor-wrap，功能强大，配置略麻烦：https://github.com/HaoChuan9421/vue-ueditor-wrap

quill-editor的Vue版本，vue-quill-editor，功能一般，配置简单：https://github.com/surmon-china/vue-quill-editor

92.是否有了解过动态表单生成概念?

所谓的动态表单就是根据数据库动态获取的字段配置信息生成表单模块，这一操作在产品的SKU管理，还有OA，ERP系统中应用广泛。

- vue-form-generator：https://github.com/vue-generators/vue-form-generator
- vue-dynamic-form-component：https://vue-dynamic-form.quincychen.cn/

93.父组件可以监听到子组件的生命周期吗

> 比如有父组件 Parent 和子组件 Child，如果父组件监听到子组件挂载 mounted 就做一些逻辑处理，可以通过以下写法实现：

```
// Parent.vue
<Child @mounted="doSomething"/>

// Child.vue
mounted() {
this.$emit("mounted");
}
复制代码
```

> 以上需要手动通过 $emit 触发父组件的事件，更简单的方式可以在父组件引用子组件时通过 @hook 来监听即可，如下所示：

```
//  Parent.vue
<Child @hook:mounted="doSomething" ></Child>

doSomething() {
console.log('父组件监听到 mounted 钩子函数 ...');
},

//  Child.vue
mounted(){
console.log('子组件触发 mounted 钩子函数 ...');
},

// 以上输出顺序为:
// 子组件触发 mounted 钩子函数 ...
// 父组件监听到 mounted 钩子函数 ...
复制代码
```

> 当然 @hook 方法不仅仅是可以监听 mounted，其它的生命周期事件，例如：created，updated 等都可以监听。

94.首屏加载性能优化中为了解决js阻塞需要实现库文件的抽离，如何实现?

> 修改vue.config.js配置，configureWebpack属性中加入externals，将对应的类库抽离，需要在index.html中加入对应的cdn文件。

```
module.exports ={
 publicPath:'./',
 configureWebpack:{
     // 此处可以配置 cdn 配置
    // 需要 在index.html 中引入 cdn 文件
     externals:{
         vue:'Vue',
         'vue-router':'VueRouter',
         vuex:'vue',
         'element-ui':'ELEMENT'
     }
 }
}
```

95.首屏性能优化一般有哪些方案?

- 异步路由加载
- 不打包库文件
- 关闭 sourcemap
- 开启 gzip 压缩
- 单独SSR页面生成

96.什么是XSRF攻击，Vue中如何做好相应的安全策略?

Cross-site request forgery跨站请求伪造，也被称为“One Click Attack”或者Session Riding，通常缩写为CSRF或者XSRF，是一种对网站的恶意利用。CSRF定义的主语是”请求“，是一种跨站的伪造的请求，指的是跨站伪造用户的请求，模拟用户的操作。

在Vue中进行vue-csrf插件的使用：https://github.com/nicita13/vue-csrf

97.电商平台购物车流程的处理方式有哪几种，操作流程与优势不足各有哪些

- 购物车数据本地缓存存储模式，单设备存储与应用，切换设备将无法同步，但性能优良
  - 以id为参数模式的操作方式
    - 用户查看产品详情，将产品添加到购物车，而购物车中生成的是一个单一数组，类似[1,2,1,1,3,4,4]
    - 1,2,3,4为产品id，出现的次数为购买的数量
    - 在购物车清单页面需要进行再次的数据请求与查询
    - 选中购物车产品等操作则增加程序复杂层度
  - 以对象为参数模式的操作方式
    - 用户查看产品详情，将产品添加到购物车，而购物车中生成的是一个对象数组，将会把产品整体对象传递并存储于数组当中
    - 可以给数组对象添加count（购买数量），selected（是否选中）等属性值
    - 在购物车清单页面不再需要进行数据请求与查询操作，减少性能开销
    - 选中购物车产品等操作简化程序复杂层度
- 购物车数据远程序服务器存储模式，与本地设备无关，切换设备也可以同步数据，请求次数多，服务器压力大，性能略低下
  - 将用户信息、产品id以及购买数量等信息作为参数进行订单接口的请求操作
  - 由服务器端进行订单内容的查询、新增、更新与删除的操作
  - 订单列表页则需要进行订单列表接口的请求获取与显示操作
  - 每个商品信息任何一次变化都需要与后台接口进行交互，http请求次数巨大

98.什么是SPU、什么是SKU?

> SPU = Standard Product Unit (标准化产品单元)
>
> SPU是商品信息聚合的最小单位，是一组可复用、易检索的标准化信息的集合，该集合描述了一个产品的特性。通俗点讲，属性值、特性相同的商品就可以称为一个SPU。
>
> SKU=stock keeping unit(库存量单位)
>
> SKU即库存进出计量的单位， 可以是以件、盒、托盘等为单位。
>
> SKU是物理上不可分割的最小存货单元。在使用时要根据不同业态，不同管理模式来处理。在服装、鞋类商品中使用最多最普遍。

99.Vue中项目中，你有了解哪些SKU的表单生成插件?

> vue-sku:基于element-ui实现zent sku商品规格组件。 https://github.com/easeava/vue-sku
>
> vue-sku-form：基于 Vue & ElementUI 的电商 SKU 表单配置组件。 https://hooray.github.io/vue-sku-form/

100.Vuex中命名空间使用后组件调用带命名空间的action方法里有几种方式?

> - 直接利用 store进行单个调用：this.$store.dispatch('moduleName/run', command);
> - 利用mapActions映射，空间模块直接映射方式
>
> ```
> mapActions([
>     'some/nested/module/foo',
>     'some/nested/module/bar'
> ])
> ```
>
> 空间模块调用方式：
>
> ```
> this['some/nested/module/foo']()
> this['some/nested/module/bar']()
> ```
>
> - 利用mapActions映射，空间模块名称+方法数组方式
>
> ```
> ...mapActions('some/nested/module', [
>     'foo',
>     'bar'
>   ])
> ```
>
> 空间模块调用方式：
>
> ```
> // -> this.foo()
> // -> this.bar()
> ```