Vue 面试题解析

序号	1
题目	uniapp 是不是基于 vue 研发 ？
解析	Uniapp 是一款基于 Vue.js 框架的跨平台开发工具，其特点是可以一次编写，多端发布，即只需要编写一套代码就可以发布到多个平台，包括微信小程序、支付宝小程序、H5、安卓 App、IOS App 等。 Uniapp 所使用的语言是 Vue.js，而 Vue.js 是当下非常流行的前端框架之一，其具有轻量、高效、易用等优点，广泛应用于各种 Web 应用开发中。Vue.js 不仅可以进行 Web 开发，还可以使用 Vue.js 的跨平台搭建框架进行移动端应用的开发。 Uniapp 结合 Vue.js 框架，为开发者提供了非常方便的开发工具和开发流程。开发者只需按照 Vue.js 的开发规范进行开发，即可轻松构建出运行在多个平台上的应用。同时，Uniapp 还提供了丰富的组件和模板，可以大大简化开发的难度和速度。 Uniapp 不仅是一款良好的 Vue.js 跨平台开发工具，还可以结合各种常见的前端框架进行使用，比如 React、AngularJS 等。因此，Uniapp 具有较强的可扩展性和适应性，可以适应不同开发者的需求。 除了其开发的方便性之外，Uniapp 还具有快速上手和快速学习的特点。对于已经熟悉 Vue.js 的前端开发者来说，上手 Uniapp 只需要很少的时间。对于初学者来说，Uniapp 提供了丰富的文档和教程，可以轻松入门。 总之，Uniapp 的开发效率和跨平台方便性，使其成为了众多开发者和企业的首选。当下，随着跨平台应用的不断发展和普及，Uniapp 也将越来越受到关注和青睐。

序号	2
题目	简述 Vue 的 MVVM 模式?
解析	MVVM 是 Model-View-ViewModel 的缩写，即将数据模型与数据表现层通过数据驱动进行分离，从而只需要关系数据模型的开发，而不需要考虑页面的表现，具体说来如下：Model 代表数据模型：主要用于定义数据和操作的业务逻辑。View 代表页面展示组件（即 dom 展现形式）：负责将数据模型转化成 UI 展现出来。ViewModel 为 model 和 view 之间的桥梁：监听模型数据的改变和控制视图行为、处理用户交互。通过双向数据绑定把 View 层和 Model 层连接了起来，而 View 和 Model 之间的同步工作完全是自动的，无需人为干涉在 MVVM 架构下，View 和 Model 之间并没有直接的联系，而是通过 ViewModel 进行交互，Model 和 ViewModel 之间的交互是双向的， 因此 View 数据的变化会同步到 Model 中，而 Model 数据的变化也会立即反应到 View 上。 1：Model（模型） 模型是指代表真实状态内容的领域模型（面向对象），或指代表内容的数据访问层（以数据为中心）。 2：View（视图） 就像在 MVC 和 MVP 模式中一样，视图是用户在屏幕上看到的结构、布局和外观（UI）。

	3：ViewModel（视图模型） 视图模型是暴露公共属性和命令的视图的抽象。MVVM 没有 MVC 模式的控制器，也没有 MVP 模式的 presenter，有的是一个绑定器。在视图模型中，绑定器在视图和数据绑定器之间进行通信。 MVVM 优点: 低耦合 :View 可以独立于 Model 变化和修改,一个 ViewModel 可以绑定到不同的 View 上,当 View 变化 的时候 Model 可以不变,当 Model 变化的时候 View 也可以不变。 可重用性 ：可以把一些视图逻辑放在一个 ViewModel 里面,让很多 View 重用这段视图逻辑。 独立开发 ：开发人员可以专注于业务逻辑和数据的开发,设计人员可以专注于页面的设计

序号	3
题目	简述 Vue 组件通讯有哪些方式 ？
解析	Vue 组件通讯有方式： 1、props 和 $emit 父组件向子组件传递数据是通过 props 传递的，子组件传递给父组件是通过$emit 触发事件来做到的。 2、$parent 和 $children 获取单签组件的父组件和当前组件的子组件。 3、$attrs 和 $listeners A -> B -> C。Vue2.4 开始提供了$attrs 和$listeners 来解决这个问题。 4、父组件中通过 provide 来提供变量，然后在子组件中通过 inject 来注入变量。（官方不推荐在实际业务中适用，但是写组件库时很常用。） 5、$refs 获取组件实例。 6、envetBus 兄弟组件数据传递，这种情况下可以使用事件总线的方式。 7、vuex 状态管理

序号	4
题目	简述 Vue 的生命周期方法有哪些？
解析	Vue 的生命周期方法： 1：beforeCreate 在实例初始化之后，数据观测（data observe）和 event/watcher 事件配置之前被调用。在当前阶段 data、methods、computed 以及 watch 上的数据和方法都不能被访问。 2：created 实例已经创建完成之后被调用。在这一步，实例已经完成以下的配置：数据观测（data observe ），属性和方法的运算，watch/event 事件回调。这里没有 $el，如果非要想与 DOM 进行交互，可以通过 vm.$nextTick 来访问 DOM。 3：beforeMount 在挂载开始之前被调用：相关的 render 函数首次被调用。 4：mounted 在挂载完成后发生，在当前阶段，真实的 Dom 挂载完毕，数据完成双向绑定，可以访问到 Dom 节点。 5：beforeUpdate 数据更新时调用，发生在虚拟 DOM 重新渲染和打补丁 （patch）之前。可以在这个钩子中进一步地更改状态，这不会触发附加的重渲染过程。（数据修改页面未修改） 6：updated 发生在更新完成之后，当前阶段组件 Dom 已经完成更新。要注意的是避免在此期间更新数据，因为这个可能导致无限循环的更新，该钩子在服务器渲染期间不被调用。 7：beforeDestroy 实例销毁之前调用。在这一步，实例仍然完全可用。我们可以在这时进行 善后收尾工作，比如清除定时器。 8：destroyed Vue 实例销毁后调用。调用后，Vue 实例指示的东西都会解绑定，所有的事件监听器会被移除，左右的子实例也会被销毁，该钩子在服务器端渲染不被调用。

	9：activated keep-alive 专属，组件被激活时调用 10：deactivated keep-alive 专属，组件被销毁时调用

序号	5
题目	简述 Vue 有哪些内置指令 ？
解析	vue 内置指令： v-once - 定义它的元素或组件只渲染一次，包括元素或组件的所有节点，首次渲染后，不再随数据的变化重新渲染，将被视为静态内容。 v-cloak - 这个指令保持在元素上直到关联实例结束编译 -- 解决初始化慢到页面闪动的最佳实践。 v-bind - 绑定属性，动态更新 HTML 元素上的属性。例如 v-bind:class。 v-on - 用于监听 DOM 事件。例如 v-on:click v-on:keyup v-html - 赋值就是变量的 innerHTML -- 注意防止 xss 攻击 v-text - 更新元素的 textContent v-model - 1、在普通标签。变成 value 和 input 的语法糖，并且会处理拼音输入法的问题。2、再组件上。也是处理 value 和 input 语法糖。 v-if / v-else / v-else-if。可以配合 template 使用；在 render 函数里面就是三元表达式。 v-show - 使用指令来实现 -- 最终会通过 display 来进行显示隐藏 v-for - 循环指令编译出来的结果是 -L 代表渲染列表。优先级比 v-if 高最好不要一起使用，尽量使用计算属性去解决。注意增加唯一 key 值，不要使用 index 作为 key。 v-pre - 跳过这个元素以及子元素的编译过程，以此来加快整个项目的编译速度。

序号	6
题目	简述怎样理解 Vue 的单项数据流 ？
解析	Vue 的单向数据流是指数据在 Vue 应用中的流动方向是单向的，数据总是从父组件传到子组件，子组件没有权利修改父组件传过来的数据，只能请求父组件对原始数据进行修改。这样会防止从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。 注意：在子组件直接用 v-model 绑定父组件传过来的 props 这样是不规范的写法，开发环境会报警告。 如果实在要改变父组件的 props 值可以再 data 里面定义一个变量，并用 prop 的值初始化它，之后用$emit 通知父组件去修改。 多种方法实现：在子组件直接用 v-model 绑定父组件传过来的 props 这种单向数据流的设计有以下几个优点： 1. 易于追踪数据流：由于数据的流动方向是单向的，我们可以很容易地追踪数据的来源和去向，减少了数据流动的复杂性，提高了代码的可读性和可维护性。 2. 提高组件的可复用性：通过 props 将数据传递给子组件，使得子组件可以独立于父组件进行开发和测试。这样一来，我们可以更方便地复用子组件，提高了组件的可复用性。 3. 避免数据的意外修改：由于子组件不能直接修改父组件的数据，可以避免数据被意外修改的情况发生。这样可以提高应用的稳定性和可靠性。 单向数据流也有一些限制和不足之处。例如，当数据需要在多个组件之间进行共享时，通过 props 传递数据会变得繁琐，这时可以考虑使用 Vuex 等状态管理工具来管理共享数据。单向数据流也可能导致

	组件之间的通信变得复杂，需要通过事件的方式进行数据传递和更新。 Vue 的单向数据流是一种有助于提高应用可维护性和可预测性的设计模式，通过明确数据的流动方向，使得代码更易于理解和维护。

序号	7
题目	简述 Vue 2.0 响应式数据的原理（ 重点 ）？
解析	整体思路是 数据劫持 + 观察者模式 Vue 在初始化数据时 ，会使用 Object.defineProperty 重新定义 data 中的所有属性 ，当页面 使用对 应 属性时，首先会进行 依赖收集 (收集当前组件的 watcher)，如果属性 发生变化 会通知相关 依赖进行 更新操作(发布订阅) Vue2.x 采用 数据劫持结合发布订阅模式 (PubSub 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。 当把一个普通 Javascript 对象传给 Vue 实例来作为它的 data 选项时，Vue 将遍历它的属性，用 Object.defineProperty 将它们转为 getter/setter。用户看不到 getter/setter，但是在内部它们让 Vue 追踪依赖，在属性被访问和修改时 通知变化 。 Vue 的数据 双向绑定 整合了 Observer，Compile 和 Watcher 三者，通过 Observer 来监听 自己的 model 的数据变化，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（例如 input 操作）->数据 model 变更的双向绑定效果。 Vue3.x 放弃了 Object.defineProperty ，使用 ES6 原生的 Proxy，来解决以前使用 Object.defineProperty 所存在的一些问题。 1、Object.defineProperty 数据劫持 2、使用 getter 收集依赖 ，setter 通知 watcher 派发更新。 3、watcher 发布订阅模式。

序号	8
题目	简述 Vue 如何检测数组变化 ？
解析	Vue2.x 中实现检测数组变化的方法，是将数组的常用方法进行了 重写 。Vue 将 data 中的数组进行了 原型链重写 ，指向了自己定义的数组原型方法。这样当调用数组 api 时，可以通知依赖更新 。如果数组中包含着引用类型，会对数组中的引用类型 再次递归遍历进行监控 。这样就实现了 监测数组变化 。 流程: 1. 初始化传入 data 数据执行 initData 2. 将数据进行观测 new Observer 3. 将数组原型方法指向重写的原型 4. 深度观察数组中的引用类型 有两种情况无法检测到数组的变化 。 1. 当利用索引直接设置一个数组项时，例如 vm.items[indexOfItem] = newValue 2. 当修改数组的长度时，例如 vm.items.length = newLength 不过这两种场景都有对应的解决方案。 利用索引设置数组项的替代方案 //使用该方法进行更新视图

	// vm.$set，Vue.set 的一个别名 vm.$set(vm.items, indexOfItem, newValue)

序号	9
题目	请解释 Vue 的父子组件生命周期钩子函数执行顺序 ？
解析	加载渲染过程 父 beforeCreate -> 父 created -> 父 beforeMount -> 子 beforeCreate -> 子 created -> 子 beforeMount -> 子 mounted -> 父 mounted 子组件更新过程 父 beforeUpdate -> 子 beforeUpdate -> 子 updated -> 父 updated 父组件更新过程 父 beforeUpdate -> 父 updated 销毁过程 父 beforeDestroy -> 子 beforeDestroy -> 子 destroyed -> 父 destroyed 总结： 父组件先开始 子组件先结束

序号	10
题目	简述 vue-router 路由钩子函数是什么？执行顺序是什么？
解析	路由钩子的执行流程，钩子函数种类有：全局守卫、路由守卫、组件守卫。 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 beforeRouterLeave 守卫。 3 、调用全局的 beforeEach 守卫。 4 、在重用的组件调用 beforeRouterUpdate 守卫 （ 2.2+ ）。 5 、在路由配置里面 beforeEnter 。 6 、解析异步路由组件。 7 、在被激活的组件里调用 beforeRouterEnter 。 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 、调用 beforeRouterEnter 守卫中传给 next 的回调函数，创建好的组件实例会作为回 调函数的参数传入。

序号	11
题目	请简述 vue-router 动态路由是什么？

解析	我们经常需要把某种模式匹配到的所有路由，全都映射到同个组件。例如，我们有一个
	User 组件，对于所有 ID 各不相同的用户，都要使用这个组件来渲染。那么，我们可以在 vue-router
	的路由路径中使用 “ 动态路径参数 ” （ dynamic segment ）来达到这个效果：
	
	const User = {
	template: "
	User", };
	const router = new VueRouter({

	routes: [// 动态路径参数 以冒号开头 { path: "/user/:id", component: User },], });

序号	12
题目	简述 vue-router 组件复用导致路由参数失效怎么办？

解析	解决方案 ：
	通 过 watch 监听 路由参数再发请求
	watch：{
	"router":function(){
	this.getData(this.$router.params.xxx)
	}
	}

序号	13
题目	Vue 生命周期钩子是如何实现的？
解析	生命周期钩子在内部会被 vue 维护成一个数组(vue 内部有一个方法 mergeOption)和全局的生命周期合并最终转换成数组，当执行到具体流程时会执行钩子(发布订阅模式)，callHook 来实现调用。 理解 vue 中模板编译原理？ 1 会将模板变成 ast 语法树(模板编译原理的核心就是 ast-> 生成代码) 2 对 ast 语法树进行优化，标记静态节点.(vue3 中模板编译做了哪些优化 patchFlag,blockTree,事件缓存，节点缓存...) 3 代码生成 拼接 render 函数字符串 + new Function + with;

序号	14
题目	请描述 Vue 的实现原理 ？
解析	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。

序号	15
题目	请简述 Vue 组件的通信（ 兄弟组件通信 ）？
解析	首先建立一个 vue 实例空白页（js 文件） import Vue from 'vue' export default new Vue() 组件 a（数据发送方）通过使用 $emit 自定义事件把数据带过去 组件 b（数据接收方）使用而通过 $on 监听自定义事件的 callback 接收数据

序号	16
题目	Vuex 页面刷新数据丢失怎么解决？
解析	需要做 vuex 数据持久化 ，一般使 用本地储存的方案 来保存数据，可以自己设计存储方案，也可以使用第三方插件。 推荐使用 vuex-persist（ 脯肉赛斯特 ）插件，它是为 Vuex 持久化储存而生的一个插件。 不需要你手动存取 storage ，而是直接将状态保存至 cookie 或者 localStorage 中

序号	17
题目	请叙述 Vue 中使用了哪些设计模式？
解析	1 工厂模式 - 传入参数即可创建实例 虚拟 DOM 根据参数的不同返回基础标签的 Vnode 和组件 Vnode。 2 单例模式 - 整个程序有且仅有一个实例 vuex 和 vue-router 的插件 3 注册方法 install 判断如果系统存在实例就直接返回掉。 3 发布-订阅模式。（vue 事件机制） 4 观察者模式。（响应式数据原理） 5 装饰器模式（@装饰器的用法） 6 策略模式，策略模式指对象有某个行为，但是在不同的场景中，该行为有不同的实现方案 - 比如选项的合并策略。

序号	18
题目	请简述 Vue 的性能优化可以从哪几个方面去思考设计 ？
解析	这里只列举针对 Vue 的性能优化，整个项目的性能优化是一个大工程。 对象层级不要过深，否则性能就会差。

	不需要响应式的数据不要放在 data 中（可以使用 Object.freeze() 冻结数据） v-if 和 v-show 区分使用场景 computed 和 watch 区分场景使用 v-for 遍历必须加 key，key 最好是 id 值，且避免同时使用 v-if 大数据列表和表格性能优化 - 虚拟列表 / 虚拟表格 防止内部泄露，组件销毁后把全局变量和时间销毁 图片懒加载 路由懒加载 异步路由 第三方插件的按需加载 适当采用 keep-alive 缓存组件 防抖、节流的运用 服务端渲染 SSR or 预渲染

序号	19
题目	简述 Vue.set 方法原理 ？
解析	Vue 响应式原理的同学都知道在两种情况下修改 Vue 是不会触发视图更新的。 在实例创建之后添加新的属性到实例上（给响应式对象新增属性） 直接更改数组下标来修改数组的值。 Vue.set 或者说是 $set 原理如下 因为响应式数据 我们给对象和数组本身新增了 __ob__ 属性，代表的是 Observer 实例。 当给对象新增不存在的属性，首先会把新的属性进行响应式跟踪 然后会触发对象 __ob__ 的 dep 收集到的 watcher 去更新，当修改数组索引时我们调用数组本身的 splice 方法去更新数组。

序号	20
题目	Vue 的组件 data 为什么必须是一个函数？
解析	new Vue 是一个单例模式，不会有任何的合并操作，所以根实例不必校验 data 一定是一个函数。 组件的 data 必须是一个函数，是为了防止两个组件的数据产生污染。 如果都是对象的话，会在合并的时候，指向同一个地址。 而如果是函数的时候，合并的时候调用，会产生两个空间。

序号	21
题目	vue 通过数据劫持可以精准的探测数据变化，为什么还要进行 diff 检测差异？
解析	现代前端框架有两种方式侦测变化,一种是 pull 一种是 push pull: 其代表为 React,我们可以回忆一下 React 是如何侦测到变化的,我们通常会用 setStateAPI 显式更新,然后 React 会进行一层层的 Virtual Dom Diff 操作找出差异,然后 Patch 到 DOM 上,React 从一开始就不知道到底是哪发生了变化,只是知道「有变化了」,然后再进行比较暴力的 Diff 操作查找「哪发生变化了」，另外一个代表就是 Angular 的脏检查操作。 push: Vue 的响应式系统则是 push 的代表,当 Vue 程序初始化的时候就会对数据 data 进行依赖的收集,一但数据发生变化,响应式系统就会立刻得知,因此 Vue 是一开始就知道是「在哪发生变化了」,但是这又会产生一个问题,如果你熟悉 Vue 的响应式系统就知道,通常一个绑定一个数据就需要一个 Watcher,一但我们的绑定细粒度过高就会产生大量的 Watcher,这会带来内存以及依赖追踪的开销,而细粒度过低

	会无法精准侦测变化,因此 Vue 的设计是选择中等细粒度的方案,在组件级别进行 push 侦测的方式,也就是那套响应式系统,通常我们会第一时间侦测到发生变化的组件,然后在组件内部进行 Virtual Dom Diff 获取更加具体的差异,而 Virtual Dom Diff 则是 pull 操作,Vue 是 push+pull 结合的方式进行变化侦测的.

序号	22
题目	请说明 Vue key 的作用及原理 ？
解析	key 是虚拟 DOM 对象的标识，当数据发生变化时，Vue 会根据[新数据]生成[新的虚拟 DOM]，随后 Vue 进行[新虚拟 DOM]与[旧虚拟 DOM]的差异比较 原理（比较规则）： 1.旧虚拟 DOM 中找到了与新虚拟 DOM 相同的 key: a.若虚拟 DOM 中内容没变，直接使用之前的直 DOM! b.若虚拟 DOM 中内容变了，则生成新的真实 DOM，随后替换掉页面中之前的真实 DOM。 2.旧虚拟 DOM 中未找到与新虚拟 DOM 相同的 key: a.创建新的真实 DOM，随后渲染到到页面。 index 作为 key 可能会引发的问题: 1.若对数据进行:逆序添加、逆序删除等破坏顺序操作: a.会产生没有必要的真实 DOM 更新==> 界面效果没问题，但效率低。 2.如果结构中还包含输入类的 DOM: a.会产生错误 DOM 更新==>界面有问题

序号	23
题目	请简述 Vue 中的 v-cloak 的理解 ？
解析	使用 v-cloak 指令设置样式，这些样式会在 Vue 实例编译结束时，从绑定的 HTML 元素上被移除。 一般用于解决网页闪屏的问题，在对一个的标签中使用 v-cloak，然后在样式中设置[v-cloak]样式,[v-cloak]需写在 link 引入的 css 中，或者写一个内联 css 样式，写在 import 引入的 css 中不起作用

序号	24
题目	简述 Vue 单页面和传统的多页面区别?
解析	单页面应用（SPA） 通俗一点说就是指只有一个主页面的应用，浏览器一开始要加载所有必须的 html, js, css。所有的页面内容都包含在这个所谓的主页面中。但在写的时候，还是会分开写（页面片段），然后在交互的时候由路由程序动态载入，单页面的页面跳转，仅刷新局部资源。多应用于 pc 端。 多页面（MPA） 指一个应用中有多个页面，页面跳转时是整页刷新 单页面的优点: 用户体验好，快，内容的改变不需要重新加载整个页面，基于这一点 spa 对服务器压力较小；前后端分离；页面效果会比较炫酷（比如切换页面内容时的专场动画）。

	单页面缺点： 不利于 seo；导航不可用，如果一定要导航需要自行实现前进、后退。（由于是单页面不能用浏览器的前进后退功能，所以需要自己建立堆栈管理）；初次加载时耗时多；页面复杂度提高很多

序号	25
题目	请描述 Vue 常用的修饰符 ？
解析	.stop：等同于 JavaScript 中的 event.stopPropagation()，防止事件冒泡； .prevent：等同于 JavaScript 中的 event.preventDefault()，防止执行预设的行为（如果事件可取消，则取消该事件，而不停止事件的进一步传播）； .capture：与事件冒泡的方向相反，事件捕获由外到内； .self：只会触发自己范围内的事件，不包含子元素； .once：只会触发一次

序号	26
题目	请简述 Vue 更新数组时触发视图更新的方法?
解析	push()；pop()；shift()；unshift()；splice()；sort()；reverse()

序号	27
题目	Vue 中 delete 和 Vue.delete 删除数组的区别 ？
解析	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

序号	28
题目	请说明 Vue 的 solt 的用法?
解析	在子组件内使用特殊的 < slot>元素就可以为这个子组件开启一个 slot（插槽），在父组件模板里，插入在子组件标签内的所有内容将替代子组件的 < slot> 标签及它的内容。 简单说来就是：在子组件内部用 标签占位，当在父组件中使用子组件的时候，我们可以在子组件中插入内容，而这些插入的内容则会替换 标签的位置。 当然：单个 solt 的时候可以不对 solt 进行命名，如果存在多个 则一个可以不命名，其他必须命名，在调用的时候指定名称的对应替换 slot，没有指定的则直接默认无名称的 solt

序号	29
题目	说明对于 Vue $emit 、$on 、$once 、$off 理解?
解析	$emit 触发当前实例上的自定义事件（并将附加参数都传给监听器回调） $on 监听实例上自定义事件并调用回调函数，监听 emit 触发的事件 $once 监听一个自定义事件，但是只触发一次，在第一次触发之后移除监听器。

	$off 用来移除自定义事件监听器。如果没有提供参数，则移除所有的事件监听器；如果只提供了事件，则移除该事件所有的监听器；如果同时提供了事件与回调，则只移除这个回调的监听器。 这四个方法的实现原理是：通过对 vue 实例挂载，然后分别使用对象存储数组对应的函数事件，其中 emit 通过循环查找存储的数组中对应的函数进行调用，once 只匹配一次就就结束，on 是将对应的函数存储到数组中，off 是删除数组中指定的元素或者所有的元素事件。具体可以参考文章：VUEemit 实现

序号	30
题目	请说明 Vue 中$root、$refs、$parent 的使用 ？
解析	$root 可以用来获取 vue 的根实例，比如在简单的项目中将公共数据放再 vue 根实例上(可以理解为一个全局 store),因此可以代替 vuex 实现状态管理； $refs 在子组件上使用 ref 特性后，this.属性可以直接访问该子组件。可以代替事件 emit 和$on 的作用。 使用方式是通过 ref 特性为这个子组件赋予一个 ID 引用，再通过 this.$refs.testId 获取指定元素。 注意：$refs 只会在组件渲染完成之后生效，并且它们不是响应式的。这仅作为一个用于直接操作子组件的“逃生舱”——你应该避免在模板或计算属性中访问$refs。 $parent $parent 属性可以用来从一个子组件访问父组件的实例，可以替代将数据以 prop 的方式传入子组件的方式；当变更父级组件的数据的时候，容易造成调试和理解难度增加；