Vue3.0面试题汇总

一、Options Api与Composition Api的区别?

1、`Options Api`：选项API，即以vue为后缀的文件，通过定义`methods`，`computed`，`watch`，`data`等属性与方法，共同处理页面逻辑。
用组件的选项(`data、computed、methods、watch`)组织逻辑在大多数情况下都有效。然而，当组件变得复杂，导致对应属性的列表也会增长，这可能会导致组件难以阅读和理解。

2、`Composition API`中，组件根据逻辑功能来组织的，一个功能所定义的所有`API`会放在一起（更加的**高内聚，低耦合**）。

3、`Composition Api`相对`Options Api`的两大优点：

- **逻辑组织**
  - `Options Api`在处理一个大型的组件时，内部的逻辑点容易碎片化，可能同时存在于`method,computed,watch`等API中，我们必须不断地“跳转”相关代码的选项块，这种碎片化使得理解和维护复杂组件变得困难。
  - `Composition Api`将某个逻辑关注点相关的代码全都放在一个函数里，这样，当需要修改一个功能时，就不再需要在文件中跳来跳去。
- **逻辑复用**
  - 在`vue2.0`中，当混入多个`mixin`会存在两个非常明显的问题：命名冲突、数据来源不清晰
  - 而`Composition Api`可以通过编写多个`hooks函数`就很好的解决了

总结

- 在逻辑组织和逻辑复用方面，`Composition API`是优于`Options API`
- 因为`Composition API`几乎是`函数`，会有更好的`类型推断`。
- `Composition API`对`tree-shaking`友好，`代码也更容易压缩`
- `Composition API`中见不到`this`的使用，减少了`this`指向不明的情况
- 如果是小型组件，可以继续使用`Options API`，也是十分友好的

二、Vue3.0性能提升主要是通过哪几方面体现的?

1、编译阶段优化

回顾`Vue2`，我们知道每个组件实例都对应一个`watcher 实例`，它会在组件渲染的过程中把用到的数据`property`记录为依赖，当依赖发生改变，触发`setter`，则会通知`watcher`，从而使关

联的组件重新渲染。

因此， Vue3 在编译阶段，做了进一步优化：

① diff算法优化

vue3 在 diff 算法中相比 vue2 增加了 静态标记 ，其作用是为了会发生变化的地方添加一个 flag标记 ，下次发生变化的时候 直接 找该地方进行比较。

② 静态提升

Vue3中对 不参与更新 的元素，会做静态提升， 只会被创建一次 ，在渲染时直接复用。免去了重复的创建操作，优化内存。

没做静态提升之前，未参与更新的元素也在 render函数 内部，会重复 创建阶段 。
做了静态提升后，未参与更新的元素，被 放置在render 函数外 ，每次渲染的时候只要 取出 即可。同时该元素会被打上 静态标记值为-1 ，特殊标志是 负整数 表示永远不会用于 Diff 。

③ 事件监听缓存

默认情况下绑定事件行为会被视为动态绑定（ 没开启事件监听器缓存 ），所以 每次 都会去追踪它的变化。 开启事件侦听器缓存 后，没有了静态标记。也就是说下次 diff算法 的时候 直接使用 。

④ SSR优化

当静态内容大到一定量级时候，会用 createStaticVNode 方法在客户端去生成一个 static node ，这些 静态node ，会被直接 innerHtml ，就不需要创建对象，然后根据对象渲染。

2、源码体积

相比 Vue2 ， Vue3 整体体积 变小 了，除了移出一些 不常用的API ，最重要的是 Tree shanking 。

任何一个函数，如 ref、reavtived、computed 等，仅仅在 用到 的时候才 打包 ， 没用到 的模块都 被摇掉 ，打包的整体体积 变小 。

3、响应式系统

vue2 中采用 defineProperty 来劫持整个对象，然后进行深度遍历所有属性，给 每个属性 添加 getter和setter ，实现响应式。

vue3 采用 proxy 重写了响应式系统，因为 proxy 可以对 整个对象进行监听 ，所以不需要深度遍历。

- 可以监听动态属性的添加
- 可以监听到数组的索引和数组length属性
- 可以监听删除属性

三、Vue3.0里为什么要用 Proxy API 替代 defineProperty API？

1、`vue2`中采用`defineProperty`来劫持整个对象，然后进行深度遍历所有属性，给每个属性添加getter和setter，实现响应式。但是存在以下的问题：

- 检测不到对象属性的添加和删除
- 数组API方法无法监听到
- 需要对每个属性进行遍历监听，如果嵌套对象，需要深层监听，造成性能问题

2、proxy：监听是针对一个对象的，那么对这个对象的所有操作会进入监听操作。

总结：

- Object.defineProperty只能遍历对象属性进行劫持
- Proxy直接可以劫持整个对象，并返回一个新对象，我们可以只操作新的对象达到响应式目的
- Proxy可以直接监听数组的变化（push、shift、splice）
- Proxy有多达13种拦截方法,不限于apply、ownKeys、deleteProperty、has等等，这是Object.defineProperty不具备的

四、Vue3.0响应式原理

vue3 响应式是使用 ES6 的 proxy 和 Reflect 相互配合实现数据响应式，解决了 vue2 中视图不能自动更新的问题。

proxy 是深度监听，所以可以监听对象和数组内的任意元素，从而可以实现视图实时更新。

详细的原理可查看vue3.0 响应式原理(超详细)

总结响应式大致分为三个阶段:

- `初始化阶段`：初始化阶段通过组件初始化方法形成对应的`proxy对象`，然后形成一个负责渲染的`effect`。
- `get依赖收集阶段`：通过`解析template`，替换`真实data`属性，来触发`get`,然后通过`stack方法`，通过`proxy对象`和`key`形成对应的`deps`，将负责渲染的`effect`存入`deps`。（这个过程还有其他的effect，比如watchEffect存入deps中）。
- `set派发更新阶段`：当我们`this[key] = value`改变属性的时候，首先通过`trigger`方法，通过`proxy对象`和`key`找到对应的`deps`，然后给`deps`分类分成`computedRunners`和`effect`,然后依次执行，如果需要`调度`的，直接放入调度。

Proxy只会代理对象的第一层，那么Vue3又是怎样处理这个问题的呢?

判断当前Reflect.get的返回值是否为Object，如果是则再通过 reactive 方法做代理， 这样就实现了深度观测。

监测数组的时候可能触发多次get/set，那么如何防止触发多次呢?

我们可以判断key是否为当前被代理对象target自身属性，也可以判断旧值与新值是否相等，只有满足以上两个条件之一时，才有可能执行trigger。

五、说说Vue 3.0中Treeshaking特性？举例说明一下？

1、是什么？

- `Tree shaking` 是一种通过 `清除多余代码` 方式来优化项目 `打包体积` 的技术，专业术语叫 `Dead code elimination`
- 简单来讲，就是在保持代码 `运行结果不变` 的前提下，去除无用的代码

在 `Vue2` 中，无论我们使用什么功能，它们最终都会出现在生产代码中。主要原因是 `Vue` 实例在项目中是单例的，捆绑程序无法检测到该对象的哪些属性在代码中被使用到。

而 `Vue3` 源码引入 `tree shaking` 特性，将全局 `API` 进行分块。如果您不使用其某些功能，它们将不会包含在您的基础包中

2、如何做？

`Tree shaking` 是基于 `ES6` 模板语法（`import` 与 `exports`），主要是借助 `ES6` 模块的 `静态编译` 思想，在 `编译时` 就能确定模块的 `依赖关系`，以及 `输入` 和 `输出` 的变量。

`Tree shaking` 无非就是做了两件事：

- 编译阶段利用 `ES6 Module` 判断哪些模块已经加载
- 判断那些模块和变量未被使用或者引用，进而删除对应代码

3、作用（好处）？

通过 `Tree shaking`， `Vue3` 给我们带来的好处是：

- 减少程序体积（更小）
- 减少程序执行时间（更快）
- 便于将来对程序架构进行优化（更友好）

以下面试题汇总自「2022」打算跳槽涨薪，必问面试题及答案——VUE3 篇

六、Vue3 新特性有哪些？

1、性能提升

- 响应式性能提升，由原来的 `Object.defineProperty` 改为基于 `ES6` 的 `Proxy`，使其速度更快

- 重写了 `Vdom` (diff算法优化，增加静态标志)
- 进行模板编译优化（静态提升，不参与更新的元素只被创建一次）
- 更加高效的组件初始化

2、更好的支持 `typeScript`

- `Vue.js 2.x` 选用 `Flow` 做类型检查，来避免一些因类型问题导致的错误，但是 Flow 对于一些复杂场景类型的检查，支持得并不好。
- `Vue.js 3.0` 抛弃了 `Flow` ，使用 `TypeScript` 重构了整个项目
- `TypeScript` 提供了更好的类型检查，能支持复杂的 `类型推断`

3、新增 `Composition API`

`Composition API` 是 `vue3` 新增的功能，比 `mixin` 更强大。它可以把各个功能模块 `独立` 开来，提高代码逻辑的可复用性，同时代码压缩性更强。

在 `Vue3` 中，定义 `methods` 、 `watch` 、 `computed` 、 `data` 数据等都放在了 `setup()` 函数中。

`setup()` 函数会在 `created()` 生命周期之前执行。执行顺序为： `beforeCreate > setup > created`

4、新增组件

- `Fragment` 不再限制 `template` 只有一个根节点。
- `Teleport` 传送门，允许我们将控制的内容传送到任意的 `DOM` 中。
- `Suspense` 等待异步组件时渲染一些额外的内容，让应用有更好的用户体验。

5、Tree-shaking：支持摇树优化

摇树优化后会将不需要的模块修剪掉，真正需要的模块打到包内。优化后的项目体积只有原来的一半，加载速度更快。

6、Custom Renderer API： 自定义渲染器

实现 DOM 的方式进行 WebGL 编程。

七、vue3 组合式API生命周期钩子函数有变化吗?

`setup` 是围绕 `beforeCreate` 和 `created` 生命周期钩子运行的，所以不需要显示的定义它们。其他的钩子都可以编写到 `setup` 内。

值得注意的是`组合式API`中的钩子函数，通过在生命周期钩子前面加上 `on` 来访问组件的生命周期钩子。需要注册，并且只能在 `setup` 期间同步使用，因为它们依赖于内部的全局状态来定位当前组件实例。

下图是选项式API 和 组合式API 生命周期钩子对比：

选项式 API	Hook inside `setup`
`beforeCreate`	Not needed*
`created`	Not needed*
`beforeMount`	`onBeforeMount`
`mounted`	`onMounted`
`beforeUpdate`	`onBeforeUpdate`
`updated`	`onUpdated`
`beforeUnmount`	`onBeforeUnmount`
`unmounted`	`onUnmounted`
`errorCaptured`	`onErrorCaptured`
`renderTracked`	`onRenderTracked`
`renderTriggered`	`onRenderTriggered`
`activated`	`onActivated`

八、watch 和 watchEffect 的区别?

`watch` 和 `watchEffect` 都是监听器， `watchEffect` 是一个副作用函数。它们之间的区别有：

- `watch` ：既要指明监视的数据源，也要指明监视的回调。
- 而 `watchEffect` 可以自动监听数据源作为依赖。不用指明监视哪个数据，监视的回调中用到哪个数据，那就监视哪个数据。
- `watch` 可以访问 `改变之前和之后` 的值， `watchEffect` 只能获取 `改变后` 的值。
- `watch` 运行的时候 `不会立即执行` ，值改变后才会执行，而 `watchEffect` 运行后可 `立即执行` 。这一点可以通过 `watch` 的配置项 `immediate` 改变。
- `watchEffect` 有点像 `computed` ：

- 但 `computed` 注重的计算出来的值（回调函数的返回值）， 所以必须要写返回值。
- 而 `watcheffect` 注重的是过程（回调函数的函数体），所以不用写返回值。

- `watch` 与 `vue2.x` 中 `watch` 配置功能一致，但也有两个小坑
  - 监视 `reactive` 定义的响应式数据时， `oldValue` 无法正确获取，强 `制开启` 了深度监视（deep配置失效）
  - 监视 `reactive` 定义的响应式数据中 `某个属性` 时， `deep配置有效` 。

```
let sum = ref(0)
let msg = ref('你好啊')
let person = reactive({
        name:'张三',
        age:18,
        job:{
                j1:{
                        salary:20
                }
        }
})

//情况1：监视ref定义的响应式数据
watch(sum,(newValue, oldValue)=>{
        console.log("sum变化了", newValue, oldValue),(immediate:true)
})
//情况2：监视多个ref定义的响应式数据
watch([sum, msg],(newValue, oldValue)=>{
        console.log("sum或msg变化了", newValue, oldValue),(immediate:true)
})
//情况3：监视reactive定义的响应式数据
//若watch监视的是reactive定义的响应式数据，则无法正确获得oldValue，且强制开启了深度监视。
watch(person,(newValue, oldValue)=>{
        console.log("person变化了", newValue, oldValue),
(immediate:true,deep:false) //此处的deep配置不再生效。
})
//情况4：监视reactive所定义的一个响应式数据中的某个属性
watch(()=>person.name,(newValue, oldValue)=>{
        console.log("person.name变化了", newValue, oldValue)
})
//情况5：监视reactive所定义的一个响应式数据中的某些属性
watch([()=>person.name, ()=>person.age],(newValue, oldValue)=>{
        console.log("person.name或person.age变化了", newValue, oldValue)
})
//特殊情况:
watch(()=>person.job,(newValue, oldValue)=>{
```

```
36         console.log("person.job变化了", newValue, oldValue)
37 }, {deep:true})
38
```

九、v-if 和 v-for 的优先级哪个高?

在 `vue2` 中 `v-for` 的优先级更高，但是在 `vue3` 中优先级改变了。 `v-if` 的优先级更高。

十、script setup 是干啥的?

`scrtpt setup` 是 `vue3` 的语法糖，简化了 `组合式 API` 的写法，并且运行性能更好。使用 `script setup` 语法糖的特点：

- 属性和方法无需返回，可以直接使用。
- 引入 `组件` 的时候，会 `自动注册` ，无需通过 `components` 手动注册。
- 使用 `defineProps` 接收父组件传递的值。
- `useAttrs` 获取属性， `useSlots` 获取插槽， `defineEmits` 获取自定义事件。
- 默认 `不会对外暴露` 任何属性，如果有需要可使用 `defineExpose` 。

十一、Vue2/Vue3组件通信方式?

Vue3通信方式：

- props
- $emit
- expose / ref
- $attrs
- v-model
- provide / inject（原理：原型链）
- Vuex/pinia
- mitt

Vue2.x 组件通信共有12种

- props
- $emit / v-on
- .sync

- v-model
- ref
- children/children / children/parent
- attrs/attrs / attrs/listeners
- provide / inject
- EventBus
- Vuex
- $root
- slot

十二、ref与reactive的区别?

ref与reactive 是 `Vue3` 新推出的主要 `API` 之一，它们主要用于响应式数据的创建。

- `template` 模板中使用的数据和方法，都需要通过 `setup` 函数 `return` 出去才可以被使用。
- `ref` 函数创建的响应式数据，在模板中可以直接被使用，在 `JS` 中需要通过 `.value` 的形式才能使用。
- `ref` 函数可以接收**原始数据类型**与**引用数据类型**。
- `reactive` 函数只能接收**引用数据类型**。
- `ref` 底层还是使用 `reactive` 来做， `ref` 是在 `reactive` 上在进行了封装，增强了其能力，使它支持了对原始数据类型的处理。
- 在 `Vue3` 中 `reactive` 能做的， `ref` 也能做， `reactive` 不能做的， `ref` 也能做。

十三、EventBus与mitt区别?

`Vue2` 中我们使用 `EventBus` 来实现跨组件之间的一些通信，它依赖于 `Vue` 自带的 $on/$ `emit/$off` 等方法，这种方式使用非常简单方便，但如果使用不当也会带来难以维护的毁灭灾难。

而 `Vue3` 中移除了这些相关方法，这意味着 `EventBus` 这种方式我们使用不了， `Vue3` 推荐尽可能使用 `props/emits` 、 `provide/inject` 、 `vuex` 等其他方式来替代。

当然，如果 `Vue3` 内部的方式无法满足你，官方建议使用一些外部的辅助库，例如：mitt。

优点

- 非常小，压缩后仅有 `200 bytes` 。
- 完整 `TS` 支持，源码由 `TS` 编码。
- 跨框架，它并不是只能用在 `Vue` 中， `React` 、 `JQ` 等框架中也可以使用。
- 使用简单，仅有 `on` 、 `emit` 、 `off` 等少量实用API。

十四、谈谈pinia?

Pinia 是 `Vue` 官方团队成员专门开发的一个全新状态管理库，并且 `Vue` 的官方状态管理库已经更改为了 `Pinia`。在 Vuex 官方仓库中也介绍说可以把 `Pinia` 当成是不同名称的 `Vuex 5`，这也意味不会再出 `5` 版本了。

优点

- 更加轻量级，压缩后提交只有 `1.6kb`。
- 完整的 `TS` 的支持， `Pinia` 源码完全由 `TS` 编码完成。
- 移除 `mutations`，只剩下 `state` 、 `actions` 、 `getters` 。
- 没有了像 `Vuex` 那样的模块镶嵌结构，它只有 `store` 概念，并支持多个 `store`，且都是互相独立隔离的。当然，你也可以手动从一个模块中导入另一个模块，来实现模块的镶嵌结构。
- 无需手动添加每个 `store`，它的模块默认情况下创建就自动注册。
- 支持服务端渲染（ `SSR` ）。
- 支持 `Vue DevTools`。
- 更友好的代码分割机制。

> `Pinia` 配套有个插件 pinia-plugin-persist进行数据持久化，否则一刷新就会造成数据丢失