



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

## 第三版：Vue 20 道

Vue 吧，我很喜欢尤大，最近刚发布了 Vue 的首部纪录片，真的很好看。

### 那你能讲一讲 MVVM 吗？

MVVM 是 Model-View-ViewModel 缩写，也就是把 MVC 中的 Controller 演变成 ViewModel。Model 层代表数据模型，View 代表 UI 组件，ViewModel 是 View 和 Model 层的桥梁，数据会绑定到 viewModel 层并自动将数据渲染到页面中，视图变化的时候会通知 viewModel 层更新数据。

### 简单说一下 Vue2.x 响应式数据原理

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)。

### 那你知道 Vue3.x 响应式数据原理吗？

(还好我有看，这个难不倒我)

Vue3.x 改用 `Proxy` 替代 `Object.defineProperty`。因为 `Proxy` 可以直接监听对象和数组的变化，并且有多达 13 种拦截方法。并且作为新标准将受到浏览器厂商重点持续的性能优化。

`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？

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

(很简单啊)

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`。

面试官抬起了头。心里暗想

(这小子还行，比上两个强，应该是多多少少看过 `Vue3` 的源码了)

## 再说一下 `vue2.x` 中如何监测数组变化

使用了函数劫持的方式，重写了数组的方法，`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。

(能问到这的面试官都比较注重深度，这些常规操作要记牢)

## `nextTick` 知道吗，实现原理是什么？

在下次 `DOM` 更新循环结束之后执行延迟回调。`nextTick` 主要使用了宏任务和微任务。根据执行环境分别尝试采用

### 1、`Promise`

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

2、 MutationObserver

3、 setImmediate

4、 如果以上都不行则采用 setTimeout

定义了一个异步方法，多次调用 nextTick 会将方法存入队列中，通过这个异步方法清空当前队列。

## 说一下 Vue 的生命周期

beforeCreate 是 new Vue() 之后触发的第一个钩子，在当前阶段 data、methods、computed 以及 watch 上的数据和方法都不能被访问。

created 在实例创建完成后发生，当前阶段已经完成了数据观测，也就是可以使用数据，更改数据，在这里更改数据不会触发 updated 函数。可以做一些初始数据的获取，在当前阶段无法与 Dom 进行交互，如果非要想，可以通过 vm.\$nextTick 来访问 Dom。

beforeMount 发生在挂载之前，在这之前 template 模板已导入渲染函数编译。而当前阶段虚拟 Dom 已经创建完成，即将开始渲染。在此时也可以对数据进行更改，不会触发 updated。

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，使用 \$refs 属性对 Dom 进行操作。

beforeUpdate 发生在更新之前，也就是响应式数据发生更新，虚拟 dom 重新渲染之前被触发，你可以在当前阶段进行更改数据，不会造成重渲染。

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

updated 发生在更新完成之后，当前阶段组件 Dom 已完成更新。要注意的是避免在此期间更改数据，因为这可能会导致无限循环的更新。

beforeDestroy 发生在实例销毁之前，在当前阶段实例完全可以被使用，我们可以在这时进行善后收尾工作，比如清除计时器。

destroyed 发生在实例销毁之后，这个时候只剩下了 dom 空壳。组件已被拆解，数据绑定被卸除，监听被移出，子实例也统统被销毁。

(关于 Vue 的生命周期详解感兴趣的也请移步我的另一篇专栏)

## 你的接口请求一般放在哪个生命周期中?

接口请求一般放在 mounted 中，但需要注意的是服务端渲染时不支持 mounted，需要放到 created 中。

## 再说一下 Computed 和 Watch

Computed 本质是一个具备缓存的 watcher，依赖的属性发生变化就会更新视图。适用于计算比较消耗性能的计算场景。当表达式过于复杂时，在模板中放入过多逻辑会让模板难以维护，可以将复杂的逻辑放入计算属性中处理。

Watch 没有缓存性，更多的是观察的作用，可以监听某些数据执行回调。当我们需要深度监听对象中的属性时，可以打开 deep: true 选项，这样便会对象中的每一项进行监听。这样会带来性能问题，优化的话可以使用字符串形式监听，如果没有写到组件中，不要忘记使用 unWatch 手动注销哦。

## 说一下 v-if 和 v-show 的区别

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

当条件不成立时，v-if 不会渲染 DOM 元素，v-show 操作的是样式(display)，切换当前 DOM 的显示和隐藏。

## 组件中的 data 为什么是一个函数？

一个组件被复用多次的话，也就会创建多个实例。本质上，这些实例用的都是同一个构造函数。如果 data 是对象的话，对象属于引用类型，会影响到所有的实例。所以为了保证组件不同的实例之间 data 不冲突，data 必须是一个函数。

## 说一下 v-model 的原理

v-model 本质就是一个语法糖，可以看成是 value + input 方法的语法糖。可以通过 model 属性的 prop 和 event 属性来进行自定义。原生的 v-model，会根据标签的不同生成不同的事件和属性。

## Vue 事件绑定原理说一下

原生事件绑定是通过 addEventListener 绑定给真实元素的，组件事件绑定是通过 Vue 自定义的 \$on 实现的。

## Vue 模版编译原理知道吗，能简单说一下吗？

简单说，Vue 的编译过程就是将 template 转化为 render 函数的过程。会经历以下阶段：

### 1、生成 AST 树

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

2、优化

3、codegen

首先解析模版,生成 AST 语法树(一种用 JavaScript 对象的形式来描述整个模板)。使用大量的正则表达式对模板进行解析,遇到标签、文本的时候都会执行对应的钩子进行相关处理。

Vue 的数据是响应式的,但其实模板中并不是所有的数据都是响应式的。有一些数据首次渲染后就不会再变化,对应的 DOM 也不会变化。那么优化过程就是深度遍历 AST 树,按照相关条件对树节点进行标记。这些被标记的节点(静态节点)我们就可以跳过对它们的比对,对运行时的模板起到很大的优化作用。

编译的最后一步是将优化后的 AST 树转换为可执行的代码。

## Vue2.x 和 Vue3.x 渲染器的 diff 算法分别说一下

简单来说,diff 算法有以下过程

- 1、同级比较,再比较子节点
- 2、先判断一方有子节点一方没有子节点的情况(如果新的 children 没有子节点,将旧的子节点移除)
- 3、比较都有子节点的情况(核心 diff)
- 4、递归比较子节点

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



正常 Diff 两个树的时间复杂度是  $O(n^3)$ ，但实际情况下我们很少会进行跨层级的移动 DOM，所以 Vue 将 Diff 进行了优化，从  $O(n^3) \rightarrow O(n)$ ，只有当新旧 children 都为多个子节点时才需要用核心的 Diff 算法进行同层级比较。

Vue2 的核心 Diff 算法采用了双端比较的算法，同时从新旧 children 的两端开始进行比较，借助 key 值找到可复用的节点，再进行相关操作。相比 React 的 Diff 算法，同样情况下可以减少移动节点次数，减少不必要的性能损耗，更加的优雅。

Vue3.x 借鉴了 [ivi](#) 算法和 [inferno](#) 算法

在创建 VNode 时就确定其类型，以及在 mount/patch 的过程中采用位运算来判断一个 VNode 的类型，在这个基础之上再配合核心的 Diff 算法，使得性能上较 Vue2.x 有了提升。(实际的实现可以结合 Vue3.x 源码看。)

该算法中还运用了动态规划的思想求解最长递归子序列。

(看到这你还会发现，框架内无处不蕴藏着数据结构和算法的魅力)

## 再说一下虚拟 Dom 以及 key 属性的作用

由于在浏览器中操作 DOM 是很昂贵的。频繁的操作 DOM，会产生一定的性能问题。这就是虚拟 Dom 的产生原因。

Vue2 的 Virtual DOM 借鉴了开源库 snabbdom 的实现。

Virtual DOM 本质就是用一个原生的 JS 对象去描述一个 DOM 节点。是对真实 DOM 的一层抽象。(也就是源码中的 VNode 类，它定义在 `src/core/vdom/vnode.js` 中。)

VirtualDOM 映射到真实 DOM 要经历 VNode 的 create、diff、patch 等阶段。

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

[key 的作用是尽可能的复用 DOM 元素。]

新旧 children 中的节点只有顺序是不同的时候，最佳的操作应该是通过移动元素的位置来达到更新的目的。

需要在新旧 children 的节点中保存映射关系，以便能够在旧 children 的节点中找到可复用的节点。key 也就是 children 中节点的唯一标识。

## keep-alive 了解吗

keep-alive 可以实现组件缓存，当组件切换时不会对当前组件进行卸载。

常用的两个属性 include/exclude，允许组件有条件的进行缓存。

两个生命周期 activated/deactivated，用来得知当前组件是否处于活跃状态。

keep-alive 的中还运用了 LRU(Least Recently Used)算法。

(又是数据结构与算法，原来算法在前端也有这么多的应用)

## Vue 中组件生命周期调用顺序说一下

组件的调用顺序都是先父后子，渲染完成的顺序是先子后父。

组件的销毁操作是先父后子，销毁完成的顺序是先子后父。

## 加载渲染过程

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题





微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

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

### Vue2.x 组件通信有哪些方式?

父子组件通信

父->子 props, 子->父 \$on、\$emit

获取父子组件实例 \$parent、\$children

Ref 获取实例的方式调用组件的属性或者方法

Provide、inject 官方不推荐使用，但是写组件库时很常用

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

## 兄弟组件通信

Event Bus 实现跨组件通信 `Vue.prototype.$bus = new Vue`

Vuex

## 跨级组件通信

Vuex

`$attrs`、`$listeners`

`Provide`、`Inject`

## SSR 了解吗?

SSR 也就是服务端渲染，也就是将 Vue 在客户端把标签渲染成 HTML 的工作放在服务端完成，然后再把 html 直接返回给客户端。

SSR 有着更好的 SEO、并且首屏加载速度更快等优点。不过它也有一些缺点，比如我们的开发条件会受到限制，服务器端渲染只支持 `beforeCreate` 和 `created` 两个钩子，当我们需要一些外部扩展库时需要特殊处理，服务端渲染应用程序也需要处于 Node.js 的运行环境。还有就是服务器会有更大的负载需求。

## 你都做过哪些 Vue 的性能优化，编码阶段

1、尽量减少 data 中的数据，data 中的数据都会增加 `getter` 和 `setter`，会收集对应的 `watcher`

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

- 2、 v-if 和 v-for 不能连用
- 3、 如果需要使用 v-for 给每项元素绑定事件时使用事件代理
- 4、 SPA 页面采用 keep-alive 缓存组件
- 5、 在更多的情况下，使用 v-if 替代 v-show
- 6、 key 保证唯一
- 7、 使用路由懒加载、异步组件
- 8、 防抖、节流
- 9、 第三方模块按需导入
- 10、 长列表滚动到可视区域动态加载
- 11、 图片懒加载

## SEO 优化

- 1、 预渲染
- 2、 服务端渲染 SSR

## 打包优化

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

- 1、 压缩代码
- 2、 Tree Shaking/Scope Hoisting
- 3、 使用 cdn 加载第三方模块
- 4、 多线程打包 happypack
- 5、 splitChunks 抽离公共文件
- 6、 sourceMap 优化

## 用户体验

- 1、 骨架屏
- 2、 PWA

还可以使用缓存(客户端缓存、服务端缓存)优化、服务端开启 gzip 压缩等。

(优化是个大工程，会涉及很多方面，这里申请另开一个专栏)

## hash 路由和 history 路由实现原理说一下

location.hash 的值实际就是 URL 中#后面的东西。

history 实际采用了 HTML5 中提供的 API 来实现，主要有 history.pushState() 和 history.replaceState()。

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题



微信搜一搜

磊哥聊编程

扫码关注



回复：面试题 获取最新版面试题

面试官拿起旁边已经凉透的咖啡，喝了一口。

(我难道问不倒这小子了么)

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题