



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：React 35 道

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？

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。

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

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`
- 2、`MutationObserver`
- 3、`setImmediate`

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

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。

(关于宏任务和微任务以及事件循环可以参考我的另两篇专栏)

(看到这你就会发现，其实问框架最终还是考验你的原生 JavaScript 功底)

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。

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。

面试官：(这小子基础还可以，接下来我得上上难度了)

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：

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、生成 AST 树

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



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

正常 Diff 两个树的时间复杂度是 $O(n^3)$ ，但实际上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)

面试官：(可以可以，看来是个苗子，不过自我介绍属实有些无聊，下一题)

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。)

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

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

「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

- 1、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。
- 2、常用的两个属性 include/exclude，允许组件有条件的进行缓存。
- 3、两个生命周期 activated/deactivated，用来得知当前组件是否处于活跃状态。
- 4、keep-alive 的中还运用了 LRU(Least Recently Used)算法。
- 5、（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

- 1、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。
- 2、组件的销毁操作是先父后子，销毁完成的顺序是先子后父。

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

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

Vue2.x 组件通信有哪些方式?

父子组件通信

- 1、父->子 props, 子->父 \$on、\$emit
- 2、获取父子组件实例 \$parent、\$children
- 3、Ref 获取实例的方式调用组件的属性或者方法
- 4、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

- 1、Event Bus 实现跨组件通信 `Vue.prototype.$bus = new Vue`
- 2、Vuex

跨级组件通信

- 1、Vuex
- 2、`$attrs`、`$listeners`
- 3、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

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、 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

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()。

面试官拿起旁边已经凉透的咖啡，喝了一口。

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

(我难道问不倒这小子了么)

setState 到底是异步还是同步?

答案：有时表现出异步,有时表现出同步

setState 只在合成事件和钩子函数中是“异步”的，在原生事件和 setTimeout 中都是同步的。

setState 的“异步”并不是说内部由异步代码实现，其实本身执行的过程和代码都是同步的，只是合成事件和钩子函数的调用顺序在更新之前，导致在合成事件和钩子函数中没法立马拿到更新后的值，形成了所谓的“异步”，当然可以通过第二个参数 setState(partialState, callback) 中的 callback 拿到更新后的结果。

setState 的批量更新优化也是建立在“异步”（合成事件、钩子函数）之上的，在原生事件和 setTimeout 中不会批量更新，在“异步”中如果对同一个值进行多次 setState，setState 的批量更新策略会对其进行覆盖，取最后一次的执行，如果是同时 setState 多个不同的值，在更新时会对其进行合并批量更新。#React 组件通信

什么是控制组件?

在 HTML 中，表单元素通常维护自己的状态，并根据用户输入进行更新。当用户提交表单时，来自上述元素的值将随表单一起发送。而 React 的工作方式则不同。包含表单的组件将跟踪其状态中的输入值，并在每次回调函数(例如 onChange)触发时重新渲染组件，因为状态被更新。以这种方式由 React 控制其值的输入表单元素称为受控组件。

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

React 组件通信如何实现?

父组件向子组件通讯: 父组件可以向子组件通过传 props 的方式, 向子组件进行通讯

子组件向父组件通讯: props+回调的方式, 父组件向子组件传递 props 进行通讯, 此 props 为作用域为父组件自身的函数, 子组件调用该函数, 将子组件想要传递的信息, 作为参数, 传递到父组件的作用域中

兄弟组件通信: 找到这两个兄弟节点共同的父节点, 结合上面两种方式由父节点转发信息进行通信

跨层级通信: Context 设计目的是为了共享那些对于一个组件树而言是“全局”的数据, 例如当前认证的用户、主题或首选语言, 对于跨越多层的全局数据通过 Context 通信再适合不过

发布订阅模式: 发布者发布事件, 订阅者监听事件并做出反应, 我们可以通过引入 event 模块进行通信
全局状态管理工具: 借助 Redux 或者 Mobx 等全局状态管理工具进行通信, 这种工具会维护一个全局状态中心 Store, 并根据不同的事件产生新的状态

React 的请求应该放在哪个生命周期中?

React 的异步请求到底应该放在哪个生命周期里, 有人认为在 componentWillMount 中可以提前进行异步请求, 避免白屏, 其实这个观点是有问题的。

由于 JavaScript 中异步事件的性质, 当您启动 API 调用时, 浏览器会在此期间返回执行其他工作。当 React 渲染一个组件时, 它不会等待

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

`componentWillMount` 它完成任何事情 - React 继续前进并继续 render, 没有办法“暂停”渲染以等待数据到达。

而且在 `componentWillMount` 请求会有一些潜在的问题, 首先, 在服务器渲染时, 如果在 `componentWillMount` 里获取数据, `fetch data` 会执行两次, 一次在服务端一次在客户端, 这造成了多余的请求, 其次, 在 React 16 进行 React Fiber 重写后, `componentWillMount` 可能在一次渲染中多次调用。

目前官方推荐的异步请求是在 `componentDidMount` 中进行。

如果有特殊需求需要提前请求, 也可以在特殊情况下在 `constructor` 中请求。

React 与 Vue 的相似之处

都使用 Virtual DOM

提供了响应式 (Reactive) 和组件化 (Composable) 的视图组件。

将注意力集中保持在核心库, 而将其他功能如路由和全局状态管理交给相关的库。

redux 的工作流程?

首先, 我们看下几个核心概念:

Store: 保存数据的地方, 你可以把它看成一个容器, 整个应用只能有一个 Store。

State: Store 对象包含所有数据, 如果想得到某个时点的数据, 就要对 Store 生成快照, 这种时点的数据集, 就叫做 State。

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

Action: State 的变化, 会导致 View 的变化。但是, 用户接触不到 State, 只能接触到 View。所以, State 的变化必须是 View 导致的。Action 就是 View 发出的通知, 表示 State 应该要发生变化了。

Action Creator: View 要发送多少种消息, 就会有多少种 Action。如果都手写, 会很麻烦, 所以我们定义一个函数来生成 Action, 这个函数就叫 Action Creator。

Reducer: Store 收到 Action 以后, 必须给出一个新的 State, 这样 View 才会发生变化。这种 State 的计算过程就叫做 Reducer。Reducer 是一个函数, 它接受 Action 和当前 State 作为参数, 返回一个新的 State。

dispatch: 是 View 发出 Action 的唯一方法。然后我们过下整个工作流程:

首先, 用户 (通过 View) 发出 Action, 发出方式就用到了 dispatch 方法。

然后, Store 自动调用 Reducer, 并且传入两个参数: 当前 State 和收到的 Action, Reducer 会返回新的 State

State 一旦有变化, Store 就会调用监听函数, 来更新 View。到这儿为止, 一次用户交互流程结束。可以看到, 在整个流程中数据都是单向流动的, 这种方式保证了流程的清晰。

redux 中间件有哪些, 做什么用?

中间件提供第三方插件的模式, 自定义拦截 action -> reducer 的过程。变为 action -> middlewares -> reducer。这种机制可以让我们改变数据流, 实现如异步 action, action 过滤, 日志输出, 异常报告等功能。常见的中间件:

redux-logger: 提供日志输出

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

redux-thunk：处理异步操作

redux-promise：处理异步操作，actionCreator 的返回值是 promise

diff 算法?

把树形结构按照层级分解，只比较同级元素。

给列表结构的每个单元添加唯一的 key 属性，方便比较。

React 只会匹配相同 class 的 component (这里面的 class 指的是组件的名字)

合并操作,调用 component 的 setState 方法的时候, React 将其标记为 dirty. 到每一个事件循环结束, React 检查所有标记 dirty 的 component 重新绘制

选择性子树渲染。开发人员可以重写 shouldComponentUpdate 提高 diff 的性能。

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题