Vue常⻅⾯试题

# Vue基础

Vue (读⾳ /vjuː/，类似于 view) 是⼀套⽤于构建⽤户界⾯的渐进式 JavaScript框架。全称是Vue.js或者Vuejs；

它基于标准 HTML、CSS 和 JavaScript 构建，并提供了⼀套声明式的、组件化的编程模型；

帮助你⾼效地开发⽤户界⾯，⽆论任务是简单还是复杂；

## Vue.js 的特点

易⽤： Vuejs是⼀个渐进式的框架，相⽐于其它框架，它更简单，易学，上⼿快。

灵活： （渐进式）不断繁荣的⽣态系统，可以在⼀个库和⼀套完整框架之间⾃如伸缩。

⾼效： 20kB min+gzip 运⾏⼤⼩；超快虚拟 DOM；最省⼼的优化。双向绑定：开发效率⾼。

基于组件的代码共享

Web项⽬⼯程化，增加可读性、可维护性

## 什么是 MVVM？

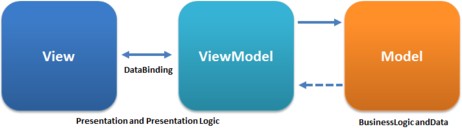
MVC和MVVM都是⼀种软件的体系结构

MVC是Model – View –Controller的简称，是在前期被使⽤⾮常框架的架构模式，⽐如iOS、前端；

MVVM是Model-View-ViewModel的简称，是⽬前⾮常流⾏的架构模式；

通常情况下，我们也经常称Vue是⼀个MVVM的框架。

Vue官⽅其实有说明，Vue虽然并没有完全遵守MVVM的模型，但是整个设计是受到它的启发的。



## 说说你对 SPA 单⻚⾯的理解，它的优缺点分别是什么？

SPA（ single-page application ）仅在 Web ⻚⾯初始化时加载相应的 HTML、JavaScript 和 CSS。⼀旦

⻚⾯加载完成，SPA 不会因为⽤户的操作⽽进⾏⻚⾯的重新加载或跳转；取⽽代之的是利⽤路由机制实现 HTML 内容的变换，UI 与⽤户的交互，避免⻚⾯的重新加载。

### 优点：

⽤户体验好、快，内容的改变不需要重新加载整个⻚⾯，避免了不必要的跳转和重复渲染；基于上⾯⼀点，SPA 相对对服务器压⼒⼩；

前后端职责分离，架构清晰，前端进⾏交互逻辑，后端负责数据处理；

### 缺点：

初次加载耗时多：为实现单⻚ Web 应⽤功能及显示效果，需要在加载⻚⾯的时候将 JavaScript、 CSS 统⼀加载，部分⻚⾯按需加载；

前进后退路由管理：由于单⻚应⽤在⼀个⻚⾯中显示所有的内容，所以不能使⽤浏览器的前进后退功能，所有的⻚⾯切换需要⾃⼰建⽴堆栈管理；

SEO 难度较⼤：由于所有的内容都在⼀个⻚⾯中动态替换显示，所以在 SEO 上其有着天然的弱势。

## v-show 与 v-if 有什么区别？

### ⾸先，在⽤法上的区别：

v-show是不⽀持template；

v-show不可以和v-else⼀起使⽤；

### 其次，本质的区别：

v-show元素⽆论是否需要显示到浏览器上，它的DOM实际都是有存在的，只是通过CSS的display

属性来进⾏切换；

v-if当条件为false时，其对应的原⽣压根不会被渲染到DOM中；

### 开发中如何进⾏选择呢？

如果我们的原⽣需要在显示和隐藏之间频繁的切换，那么使⽤v-show；

□ 如果不会频繁的发⽣切换，那么使⽤v-if；

## 数组中的哪些⽅法会触发视图的更新？

Vue 将被侦听的数组的变更⽅法进⾏了包裹，所以它们也将会触发视图更新，这些被包裹过的⽅法包括：

push()

pop() shift() unshift() splice() sort() reverse()

上⾯的⽅法会直接修改原来的数组，所以它们会触发视图更新。

其它数组的⽅法:

但是某些⽅法不会替换原来的数组，⽽是会⽣成新的数组，⽐如 ﬁlter()、concat() 和 slice()，使⽤这些⽅法将不会触发视图更新。

## Vue中v-for的key 有什么作⽤？

在使⽤v-for进⾏列表渲染时，我们通常会给元素或者组件绑定⼀个key属性。这个key属性有什么作⽤呢？

key属性主要⽤在Vue的虚拟DOM算法，在新旧nodes对⽐时辨识VNodes。

如果不使⽤key，Vue会使⽤⼀种最⼤限度减少动态元素并且尽可能的尝试就地修改/复⽤相同类型元素的算法

使⽤key时，它会基于key的变化重新排列元素顺序，并且会移除/销毁key不存在的元素。

key 是 VNode 的唯⼀标记，通过这个 key， diﬀ 操作可以更准确、更快速的达到复⽤节点，更新视图的

⽬的。复⽤节点就需要通过移动元素的位置来达到更新的⽬的。

## computed和method有什么区别？

计算属性和⽅法：

都可以通过this来访问

都可以对⼀些数据进⾏处理和计算

对于包含响应式数据计算的逻辑，应该使⽤计算属性，因为计算属性是有缓存。

computed和method的区别

computed底层会缓存, 性能更⾼

计算属性会基于它们的依赖关系进⾏缓存;

在数据不发⽣变化时，计算属性是不需要重新计算的

但是如果依赖的数据发⽣变化，在使⽤时，计算属性依然会重新进⾏计算

## 什么是双向绑定？v-model的本质是什么？

双向绑定:

即当数据发⽣变化的时候，视图也就发⽣变化，当视图发⽣变化的时候，数据也会跟着同步变化

v-model 是语法糖，它负责监听⽤户在表单元素中的输⼊事件来更新数据

### 表单元素使⽤v-model的本质：

v-bind绑定value属性的值

v-on绑定input事件监听到函数,函数会获取最新的值赋值到绑定的属性中

<input type="text" :value="message" @input="message = $event.target.value" />

### 组件使⽤v-model的本质：

将其 value attribute 绑定到⼀个名叫 modelValue 的 prop 上；

在其 input 事件被触发时，将新的值通过⾃定义的 update:modelValue 事件抛出(发出)；

<Counter v-model="appCounter"/>

<!-- 相当于-->

<Counter v-bind:modelValue="appCounter" @update:modelValue="appCounter =

$event"/>

## data选项为什么是⼀个函数⽽不是对象？

JavaScript中的对象是引⽤类型的数据，当多个实例引⽤同⼀个对象时，只要⼀个实例对这个对象进⾏操作，其他实例中的数据也会发⽣变化。

⽽在Vue中，我们更多的是想要复⽤组件，那就需要每个组件都有⾃⼰的数据，这样组件之间才不会相互⼲扰。

所以组件的数据不能写成对象的形式，⽽是要写成函数的形式。数据以函数返回值的形式定义。

这样当我们每次复⽤组件的时候，就会返回⼀个新的data，也就是说每个组件都有⾃⼰的私有数据空间，它们各⾃维护⾃⼰的数据，不会⼲扰其他组件的正常运⾏。

## Vue data 中某⼀个属性的值发⽣改变后，视图会⽴即同步执⾏重新渲染吗？

不会⽴即同步执⾏重新渲染。

Vue 实现响应式并不是数据发⽣变化之后 DOM ⽴即变化，⽽是按⼀定的策略进⾏ DOM 的更新。

Vue 在更新 DOM 时是异步执⾏的。只要侦听到数据变化， Vue 将开启⼀个队列，并缓冲在同⼀事件循环中发⽣的所有数据变更。

如果同⼀个watcher被多次触发，只会被推⼊到队列中⼀次。这种在缓冲时去除重复数据对于避免不必要的计算和 DOM 操作是⾮常重要的。

然后，在下⼀个的事件循环”tick”中，Vue 刷新队列并执⾏实际（已去重的）⼯作。

## sass是什么？如何在vue中安装和使⽤？

sass是⼀种CSS预处理器语⾔，除此之外，less、stylus也是常⻅的CSS预处理器语⾔。 sass安装和使⽤步骤如下：

* + 1. ⽤npm安装加载程序（ sass-loader、 css-loader等加载程序)。
    2. 在 webpack.conﬁg.js中配置sass加载程序。

## 在 Vue. js开发环境下调⽤API接⼝，如何避免跨域

1.在vue.conﬁg.js中的devServer选项中的proxy中配置反向代理 2.在vite.conﬁg.js中的server选项中的proxy中配置反向代理

3.直接后端开发⼈员配置cors

## v-if和v-for⼀起使⽤的弊端及解决办法

Vue.js 中使⽤最多的两个指令就是

和 v-for ，因此开发者们可能会想要同时使⽤它们。虽然不

建议这样做，但有时确实是必须的，于是我们想提供有关其⼯作⽅式的指南。

v-if

2..x 版本中在⼀个元素上同时使⽤ 和

v-if

v-if

v-for

时， v-for 会优先作⽤。

3.x 版本中

v-for

总是优先于

⽣效。

由于语法上存在歧义，建议避免在同⼀元素上同时使⽤两者。

⽐起在模板层⾯管理相关逻辑，更好的办法是通过创建计算属性筛选出列表，并以此创建可⻅元素，⽐如：

* + 1. 在v-for的外层或内层包裹⼀个元素（template）来使⽤v-if
    2. ⽤computed处理筛选出列表

## 谈谈你对 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，避免重新渲染 ，其有以下特性：

⼀般结合路由和动态组件⼀起使⽤，⽤于缓存组件。

提供 include 和 exclude 属性，两者都⽀持字符串或正则表达式。

include 表示只有名称匹配的组件会被缓存。 exclude 表示任何名称匹配的组件都不会被缓存。其中 exclude 的优先级⽐ include ⾼。

对应两个钩⼦函数 activated 和 deactivated 。

当组件被激活时，触发钩⼦函数 activated。 当组件被移除时，触发钩⼦函数 deactivated。

## 说说Vue插槽的作⽤和平时开发中的应⽤？

插槽的作⽤:

⽀持在⽗组件⾃定义⼦组件中的个内容

让⼦组件更具有通⽤性，不必限定死某个内容

插槽平时开发中的应⽤:

在封装组件时，如果组件中的某个内容是动态的或不确定的，就可以使⽤插槽来代替了。在使⽤第三⽅库时，往往会通过使⽤插槽类⾃定义第三⽅组件中的某些内容。

# Component组件

## ⽗⼦组件的⽣命周期顺序

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

## 组件通讯(传值)式有哪些?

⽗传⼦：⼦组件通过props来接收⽗组件传递的属性 xxx 的值

⼦传⽗：⼦组件通过emit触发事件传递，⽗组件通过监听对应的事件来接收数据 Provide/Inject：⽗组件提供内容，⼦或孙组件可以注⼊⽗组件提供的内容。 组件实例：通过ref来拿到组件的实例，调⽤实例的属性或⽅法进⾏传值。

事件总线：可以⾃⼰编写EventBus插件来进⾏通讯，或世界使⽤第三⽅的事件总线库。

⽤Vuex/Pinia: 可以使⽤全局状态管理来进⾏全局共享数据。

## 什么是⽣命周期函数？Vue组件的⽣命周期函数有哪些？

### ⽣命周期函数：

⽣命周期函数是⼀些钩⼦函数（回调函数），在某个时间会被Vue源码内部进⾏回调通过对⽣命周期函数的回调，我们可以知道⽬前组件正在经历什么阶段

### Vue2的⽣命周期函数:

beforeCreate :组件实例在创建之前 created: 组件被创建完成

可以发送⽹络请求

可以事件监听

this.$watch()

beforeMount : 组件template准备被挂载

mounted :组件template已经被挂载可以获取DOM,可以使⽤DOM

beforeUpdate: 准备更新DOM

updated: 更新DOM,根据最新数据⽣成新的VNode,⽣成新的虚拟DOM,转换为真实的DOM

beforeUnmount: 卸载之前

unmounted: DOM 元素被卸载完成回收操作(取消事件监听)

**Vue3的⽣命周期函数:**

# Composition API

## 什么是Composition API 和 Options API？

Composition API:

Composition API 是⼀组 API，允许我们使⽤导⼊的函数⽽不是声明选项来编写 Vue 组件。它是⼀个涵盖以下 API 的总称：Reactivity API、Lifecycle Hooks、Dependency Injection等等

使⽤Composition API编写组件时可以根据逻辑功能来组织代码。⽐如可以把⼀个功能所⽤到的 API 放在⼀起，这样可以让代码⾼内聚和低耦合，进⽽提⾼了代码的逻辑的复⽤性。

在 Vue 3 中，它也主要与script setup语法⼀起使⽤。

Options API:

在对应的属性中编写对应的功能模块, ⽐如data定义数据、methods中定义⽅法、computed中定义计算属性、watch中监听属性改变，也包括⽣命周期钩⼦

弊端: 当我们实现某⼀个功能时，这个功能对应的代码逻辑会被拆分到各个属性中,当组件变得复杂，导致对应属性的列表也会增⻓，这可能会导致组件难以阅读和理解

## Composition API和之Options API有什么区别?

在逻辑组织和逻辑复⽤⽅⾯，Composition API是优于Options API。 Composition API⼏乎是函数，会有更好的类型推断，对于TS的⽀持更友好。 Composition API对 tree-shaking 友好，代码也更容易压缩。 Composition API中⻅不到this的使⽤，减少了this指向不明的情况。

Composition API⽤起来稍微复杂⼀点，⽽Options API就⾮常简单、易于使⽤。

## 说说Vue3中setup函数的作⽤？

在Vue3中， setup() 函数充当了组件编写Composition API 的⼊⼝点。

setup函数参数主要有两个参数：

第⼀个参数：props , ⽗组件传递过来的属性会被放到props对象中第⼆个参数：context, 它⾥⾯包含三个属性

attrs：所有的⾮prop的attribute； slots：⽗组件传递过来的插槽；

emit：当我们组件内部需要发出事件时会⽤到emit（因为我们不能访问this，所以不可以通过 this.$emit发出事件）

可以在setup中可以定义响应式数据、⽅法、计算属性、侦听器等等。

可以通过setup的返回值来替代data选项，让数据可以直接在template中使⽤。

## ref和reactive有什么区别？开发中如何选择？

ref和reactive都是响应式的API，都可以⽤来定义响应式的数据。

ref可以包裹任意数据类型，reactive只能包裹复杂数据类型，⽐如对象、数组。

ref返回⼀个ref对象，在script中取值需要通过value属性，但是在模板中使⽤会进⾏解包不需要调

⽤value。

reactive包裹的是复杂数据类型，直接取⾥⾯的属性即可。

ref⼏乎可以应⽤在任何场景，⽽且包含reactive适合的场景

reactive的应⽤场景⽐较受限，第⼀：值⽐较固定，第⼆：值与值之间是有联系的。开发中尽量选择ref

## Composition API常⻅的⼏个函数与⽤法？

ref

包裹任意类型的值，将包裹的值加⼊响应式

reactive

包裹复杂类型的值，将包裹的值加⼊响应式

computed

把⼀些复杂逻辑⽤computed进⾏包裹，如同Options API中的计算属性⼀样 computed会⾃动收集相关依赖，当依赖发⽣变化时，会⾃动进⾏更新

⽣命周期

Vue3中想要在beforeCreate和created中做的事，直接在setup中做即可 Vue3的其他的⽣命周期函数都要在前⾯加⼀个on，然后需要在vue中主动引⼊

watch

watch可以监听单个数据源，也可以监听多个数据源

watch是懒执⾏，第⼀次是不会执⾏的，除⾮你为其提供第三个参数中的immediate属性为

true

watch只有等到监听的数据源发⽣了变化后，才会执⾏第⼆个参数（回调） watch可以获取监听数据源的前后变化的值

侦听多个数据源的时候，第⼀个参数是数组类型

watchEﬀect

watchEﬀect会⾃动收集依赖，收集的依赖是第⼀个参数，也就是回调函数中有哪些东⻄是加

⼊响应式的

如果这个值加⼊了响应式就会被收集起来，当被收集的值发⽣了变化，就会重新执⾏这个回调函数

watchEﬀect第⼀次执⾏是在DOM挂载前执⾏的，所以如果你想在第⼀次执⾏时拿到DOM元素

需要传⼊第⼆个参数，第⼆个参数是⼀个对象，让其ﬂush属性的值为post即可

toRefs

对reactive进⾏解构后就失去了响应式的效果，因为reactive返回的是⼀个Proxy对象对Proxy对象进⾏解构，拿到的是纯净的值，所以没有了响应式的效果

如果想要对reactive进⾏解构，需要对其包裹⼀个toRefs这么做相当于为reactive中的每⼀个值包裹了⼀个ref

等等

## Vue3中的watch和watchEﬀect有什么区别？

watch和watchEﬀect都⽤⽤来侦听响应式数据的变化，watch可以侦听指定的源，默认第⼀次不会执⾏，watchEﬀect虽不能指定侦听的源，但是会⾃动收集依赖，并默认会先执⾏⼀次。

watch

watch可以监听单个数据源，也可以监听多个数据源

watch是懒执⾏，第⼀次是不会执⾏的，除⾮你为其提供第三个参数中的immediate属性为

true

watch只有等到监听的数据源发⽣了变化后，才会执⾏第⼆个参数（回调） watch可以获取监听数据源的前后变化的值

侦听多个数据源的时候，第⼀个参数是数组类型

watchEﬀect

watchEﬀect会⾃动收集依赖，收集的依赖是第⼀个参数，也就是回调函数中有哪些东⻄是加

⼊响应式的

如果这个值加⼊了响应式就会被收集起来，当被收集的值发⽣了变化，就会重新执⾏这个回调函数

watchEﬀect第⼀次执⾏是在DOM挂载前执⾏的，所以如果你想在第⼀次执⾏时拿到DOM元素

需要传⼊第⼆个参数，第⼆个参数是⼀个对象，让其ﬂush属性的值为post即可

## 说说Vue3中script setup语法糖常⻅⽤法？

script setup 是在单⽂件组件中使⽤ Composition API 的编译时语法糖，相⽐与之前的setup函数写法，它具有更多的优势：

更少的样板内容，更简洁的代码。

能够使⽤纯 TypeScript 声明 props 和抛出事件。

更好的运⾏时性能 (其模板会被编译成与其同⼀作⽤域的渲染函数，没有任何的中间代理)。更好的 IDE 类型推断性能 (减少语⾔服务器从代码中抽离类型的⼯作)。

### script setup

当使⽤ script setup 的时候，任何在 script setup 声明的顶层绑定都能在模板中直接使⽤声明的顶层绑定：包括变量，函数声明，以及 import 引⼊的内容

响应式数据需要通过ref、reactive来创建

在script setup中导⼊的组件可以直接使⽤

### deﬁneProps

在script setup语法糖中必须使⽤ deﬁneProps API来声明props，它具备完整的类型推断并且在中是直接可⽤的（不需要额外导⼊）。

<script setup>

### deﬁneEmits

在script setup语法糖中必须使⽤ deﬁneEmits API来声明 emits，它具备完整的类型推断并且在中是直接可⽤的（不需要额外导⼊）。

<script setup>

### deﬁneExpose

获取组件的实例可以通过ref来获取，接着组件挂载完成后可通过value拿到组件实例。

当拿到组件实例后，默认是不可以访问这个实例中的⽅法和属性，因为默认没暴露任何⽅法和属性。

因此在Vue3组件中可以⽤deﬁneExpose API来暴露⽅法和属性给外部访问。 deﬁneExpose 也是不需要导⼊，直接使⽤即可

# Vue Rouer路由

## vue-router路由的两种模式

vue-router中默认使⽤的是hash模式

* + 1. hash模式，带#。如：http://localhost:8080/#/page。改变hash，浏览器本身不会有任何请求服务器动作。
    2. history模式，不带#， 如：<http://localhost:8080/page ，路径没有#。基于HTML5的

pushState、replaceState实现。

|  |  |
| --- | --- |
| **hash** | **history** |
| 有 # 号 | 没有 # 号 |
| 能够兼容到IE8 | 只能兼容到IE10 |
| 实际的url之前使⽤哈希字符，这部分url不会发送到服务器，不需要在服务器层⾯上进⾏任何处理 | 每访问⼀个⻚⾯都需要服务器进⾏路由匹配⽣成 html ⽂件再发送响应给浏览器，消耗服务器⼤量资源 |
| 刷新不会存在 404 问题 | 浏览器直接访问嵌套路由时，会报 404 问题。 |
| 不需要服务器任何配置 | 需要在服务器配置⼀个回调路由 |

## 路由跳转时如何传递数据？

动态路由

path:

/user/:id

获取动态路由的值的⽅式

在template中，直接通过 $route.params获取值在created中，通过 this.$route.params获取值

在setup中，使⽤ vue-router库提供的⼀个hook useRoute

该Hook会返回⼀个Route对象，对象中保存着当前路由相关的值

query参数:

通过query的⽅式来传递参数

在界⾯中通过 $route.query 来获取参数

在created中，通过 this.$route.query获取值

在setup，使⽤ vue-router库提供的⼀个hook useRoute 来获取

## 什么是路由守卫？路由守卫有什么作⽤？

### 什么是路由守卫：

vue-router 提供的路由(导航)守卫主要⽤来通过跳转或取消的⽅式守卫导航。有很多种⽅式植⼊路由导航中：全局的，单个路由独享的，或者组件级的。

全局导航钩⼦

router.beforeEach(to,from,next) router.afterEach(to,from)

….

组件内的钩⼦

beforeRouteEnter (to, from, next) beforeRouteUpdate (to, from, next) beforeRouteLeave (to, from, next)

….

单独路由独享组件

beforeEnter: (to, from, next) afterEach(to,from)

….

### 路由守卫有什么作⽤：

可以在进⼊路由之前进⾏某些判断，⽐如，检查token是否存在来判断⽤户是否已经登录。可以在路由守卫中进⾏⻚⾯的权限判断，⽐如，判断某个⽤户是否拥有该⻚⾯的权限。 也可以⽤来记录⻚⾯的某些信息，⽐如，记录⻚⾯的滚动信息等等。

## route和router的区别

route是路由信息对象，在Vue3中通过 useRoute 来获取。

包括了path，params，hash，query，fullPath，matched，name等路由信息参数。

router是路由实例”对象，在Vue3中通过 useRouter 来获取。

包括了路由跳转⽅法、钩⼦函数等，⽐如：push、go、back、addRouter、beforeEnter等。

# Vuex和Pinia状态管理

## 什么是状态管理？什么是单项数据流？

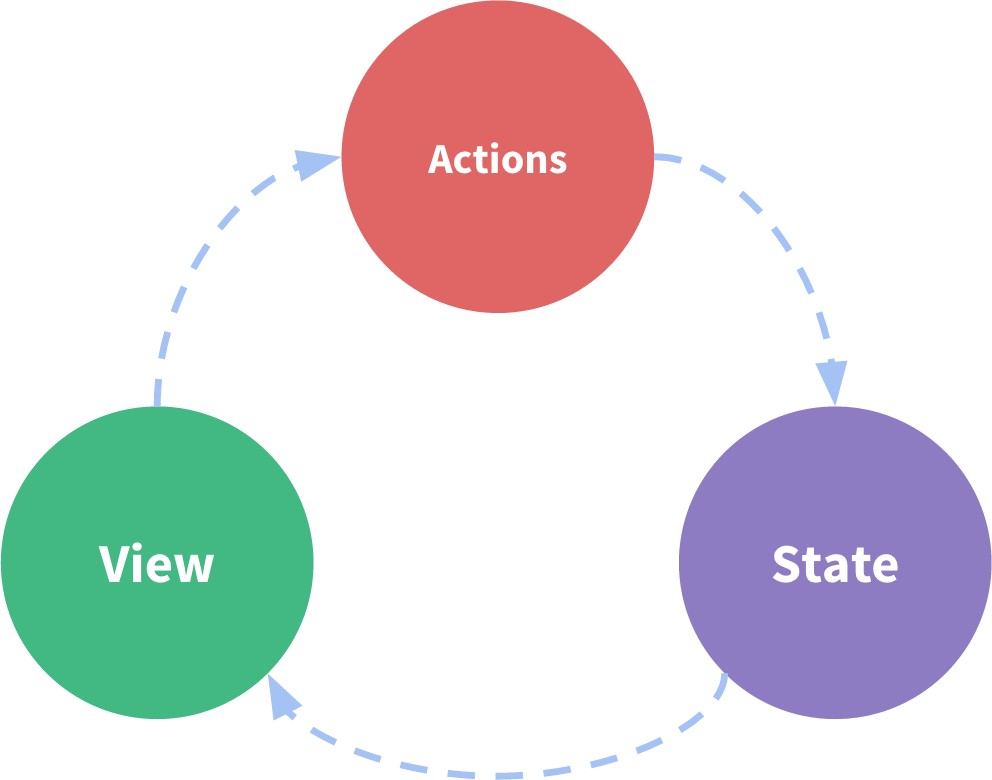
在开发中，应⽤程序是需要处理各种各样的数据，这些数据需要保存在应⽤程序中的某⼀个位置，对于这些数据的管理就称之为是 **状态管理**。以前我们是如何管理应⽤程序的状态？

在Vue开发中，我们使⽤组件化的开发⽅式。⽽在组件中我们定义的data或在setup中返回的数据，这些数据我们称之为状态（State）。

在模块template中我们可以使⽤这些数据，模块最终会被渲染成DOM，我们称之为View。

在模块中我们会产⽣⼀些⾏为事件，处理这些⾏为事件时，有可能会修改State，这些⾏为事件我们称之为Actions。

其实Vue组件内部的数据是以单向数据流的形式来管理数据的。例如，组件的数据定义在State中，接着在View层使⽤State中的数据，然后View层会产⽣⼀些事件Actions，⽽这些Actions可能会修改State的数据，这就是⼀个单项数据流的概念。



## 什么是Vuex？你使⽤过 Vuex 吗？

### 什么是Vuex：

Vuex 是⼀个专为 Vue.js 应⽤程序开发的状态管理模式。每⼀个 Vuex 应⽤的核⼼就是 store（仓库）。 “store” 基本上就是⼀个容器，它包含着你的应⽤中⼤部分的状态 ( state )。

Vuex 的状态存储是响应式的。当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发⽣变化，那么相应的组件也会相应地得到⾼效更新。

改变 store 中的状态的唯⼀途径就是显式地提交 (commit) mutation。这样使得我们可以⽅便地跟踪每⼀个状态的变化。

### Vuex包括⼀下⼏个核⼼模块：

State：定义了应⽤状态的数据结构，可以在这⾥设置默认的初始状态。

Getter：允许组件从 Store 中获取数据，mapGetters 辅助函数仅仅是将 store 中的 getter 映射到计算属性。

Mutation：是唯⼀更改 store 中状态的⽅法，且必须是同步函数。 Action：⽤于提交 mutation，⽽不是直接变更状态，可以包含任意异步操作。 Module：允许将单⼀的 Store 拆分为多个 store 且同时保存在单⼀的状态树中。

## 在哪些场景中会使⽤到Vuex？

对于简单的项⽬，是可以不⽤使⽤ Vuex，但是对于⽐较复杂，组件⽐较多的⼤项⽬就需要使⽤Vuex

了。

可⽤于记录收藏、购物⻋、应⽤配置信息、⽤户信息等场景中。

可⽤于记录系统的登录状态、⽤户权限、部⻔信息、系统配置信息等场景中。可⽤于记录城市列表数据、全局枚举、当前坐标等场景中。

⻚⾯中的组件件嵌套太深了，导致⼀层层传递数据变的⾮常麻烦了，也可以将⻚⾯数据存到Vuex中。

## 什么是Pinia？

### 什么是Pinia:

Pinia 是 Vue 的存储库，它允许您跨组件/⻚⾯共享状态。 Pinia适⽤于Vue2和Vue3，并不需要使⽤ Composition API。 Pinia的处理安装之后，它的API也同样适⽤于SSR的应⽤程序。

### Pinia⼏个核⼼概念：

state

state是⼀个选项，这个选项的值需要是⼀个函数，函数返回⼀个对象，对象中存储数据在组件中拿到当前的store直接使⽤即可，store.xxx

getters

getters也是⼀个选项，这个选项的值是⼀个对象，对象中存储着⼀个个函数，每个函数可以有⼀个参数state，通过state可以获取到当前store的state

除此之外每个函数还可以拿到⼀个this，这个this就是当前的整个store实例通过这个this，可以想⽤谁就⽤谁

在组件中使⽤也是拿到store直接store.xxx即可

actions

在actions中，主要存放⼀个个函数，每个函数最主要的⼯作发送异步请求，获取到数据后直接修改state

每个action函数并不像getter函数⼀样，第⼀个参数是state，它可以没有参数需要通过this拿到state然后再修改state中的值

在组件中拿到store后直接调⽤即可，store.xxx()

如果你在此时传递参数，那么就可以在action中拿到参数没有模块modules的概念。

## Pinia 和 Vuex有什么区别？

Pinia没有Vuex中的mutations选项，因为mutations的出现解决的问题是让devtools进⾏状态追踪但是随着技术的发展，Pinia已经解决的这个没有mutation依然可以跟踪状态的问题。

Pinia可以在任意组件中拿到store然后直接修改state中的任意值

Pinia不再需要Vuex中的modules这样的嵌套结构，取⽽代之的是可以创建⼀个个store

使⽤上的区别

在Vuex中使⽤某个state时，需要$store.state.xxx

在Pinia中直接拿到store之后store.xxx即可

在Vuex中使⽤某个getter函数时，需要$store.getters.xxx

在Pinia中拿到store后，store.xxx即可在Vuex中进⾏异步请求需要派发action函数

在Pinia中拿到store后，直接调⽤action函数即可

…...

# Vue其它问题

## 什么是虚拟DOM？什么是diﬀ算法？

### 什么是虚拟DOM：

Virtual DOM 本质上是 JavaScript 对象，是真实 DOM 的描述，⽤⼀个 JS 对象来描述⼀个 DOM 节点。

Virtual DOM 可以看做⼀棵模拟 DOM 树的 JavaScript 树，其主要是通过 VNode 实现⼀个⽆状态的组件，当组件状态发⽣更新时，然后触发 Virtual DOM 数据的变化，然后通过 Virtual DOM 和真实 DOM的⽐对，再对真实 DOM 更新。可以简单认为 Virtual DOM 是真实 DOM 的缓存。

### 虚拟 DOM 的优缺点:

优点：

**跨平台与分层设计**(主要原因)：虚拟 DOM 本质上是 JavaScript 对象，⽽真实 DOM 与平台强相关，相⽐之下虚拟 DOM 带来了分层设计、跨平台以及 SSR 等特性。⾄于 Virtual DOM ⽐原⽣ DOM 谁的性能好，需要 “控制变量法” 才能⽐较。这是为什么要设计虚拟 DOM 的主要原因。虚拟 DOM 抽象了原本的渲染过程，实现了跨平台的能⼒，⽽不仅仅局限于浏览器的 DOM，可以是安卓和 iOS 的原⽣组件，也可以是⼩程序，也可以是各种 GUI。

**以最⼩的代价更新变化的视图**。整棵 DOM 树实现代价太⾼，能否只更新变化的部分的视图。虚拟 DOM 能通过 patch 准确地转换为真实 DOM，并且⽅便进⾏ diﬀ。

**保证性能下限**(次要原因)：框架的虚拟 DOM 需要适配任何上层 API 可能产⽣的操作(分层设计)，它的⼀些 DOM 操作的实现必须是普适的，所以它的性能并不是最优的；但是⽐起粗暴

的 DOM 操作性能要好很多，因此框架的虚拟 DOM ⾄少可以保证在你不需要⼿动优化的情况下，依然可以提供还不错的性能，即保证性能的下限。

**⽆需⼿动操作 DOM**：操作 DOM 慢，js 运⾏效率⾼。我们可以将 DOM 对⽐(diﬀ 操作)放在 JS 层，提⾼效率。我们不再需要⼿动去操作 DOM，只需要写好 View-Model 的代码逻辑，框架会根据虚拟 DOM 和 数据双向绑定，帮我们以可预期的⽅式更新视图，极⼤提⾼我们的开发效率。

**组件的⾼度抽象化**：Vue.2x 引⼊ VirtualDOM 把渲染过程抽象化，从⽽使得组件的抽象能⼒也得到提升，并且可以适配 DOM 以外的渲染⽬标。不再依赖 HTML 解析器进⾏模版解析，可以进⾏更多的 AOT ⼯作提⾼运⾏时效率：通过模版 AOT 编译，Vue 的运⾏时体积可以进

⼀步压缩，运⾏时效率可以进⼀步提升。**Virtual DOM 的优势不在于单次的操作，⽽是在⼤量、频繁的数据更新下，能够对视图进⾏合理、⾼效的更新**。为了实现⾼效的 DOM 操作，

⼀套⾼效的虚拟 DOM diﬀ 算法显得很有必要.

缺点:

⽆法进⾏极致优化： 虽然虚拟 DOM + 合理的优化，⾜以应对绝⼤部分应⽤的性能需求，但在⼀些性能要求极⾼的应⽤中虚拟 DOM ⽆法进⾏针对性的极致优化。

虽然 Vue 能够保证触发更新的组件最⼩化，但在单个组件内部依然需要遍历该组件的整个

Virtual DOM 树。

在⼀些组件整个模版内只有少量动态节点的情况下，这些遍历都是性能的浪费。传统 Virtual DOM 的性能跟模版⼤⼩正相关，跟动态节点的数量⽆关。

### 什么是diﬀ算法：

diﬀ 算法是⼀种通过同层的树节点进⾏⽐较的⾼效算法。diﬀ 整体策略为：深度优先，同层⽐较。

新旧两个 VNode 节点的左右头尾两侧均有⼀个变量标识，在遍历过程中这⼏个变量都会向中间靠拢。

当 或者 时结束循环。在遍历中，如果

oldStartIdx <= oldEndIdx

newStartIdx <= newEndIdx

存在 key，并且满⾜ sameVnode，会将该 DOM 节点进⾏复⽤(只通过移动节点顺序)，否则则会创建⼀个新的 DOM 节点。

与

oldStartVnode、oldEndVnode

newStartVnode、newEndVnode

* + 1. 当新旧⼦树的两个开始节点或新旧⼦树的两个结束节点相同时

两两⽐较共有四种⽐较⽅法：

当新旧 VNode 节点的 start 或者 end 满⾜ sameVnode 时，也就是 sameVnode(oldStartVnode, newStartVnode) 或者 sameVnode(oldEndVnode, newEndVnode) 表示为 true，直接将该 VNode 节点进⾏ patchVnode 即可（保留）。

* + 1. 当旧⼦树的开始节点与新⼦树的结束节点相同时

当 oldStartVnode 与 newEndVnode 满⾜ sameVnode，即 sameVnode(oldStartVnode, newEndVnode)。这时候说明 oldStartVnode 已经跑到了 oldEndVnode 后⾯去了，进⾏ patchVnode 的同时还需要将真实 DOM 节点移动到 oldEndVnode 的后⾯。

* + 1. 当旧⼦树的结束节点与新⼦树的开始节点相同时

如果 oldEndVnode 与 newStartVnode 满⾜ sameVnode，即 sameVnode(oldEndVnode, newStartVnode)。这说明 oldEndVnode 跑到了 oldStartVnode 的前⾯，进⾏ patchVnode 的同时真实的 DOM 节点移动到了 oldStartVnode 的前⾯。

* + 1. 当旧⼦树中没有新⼦树中的节点，会将新节点插⼊到 oldStartVnode 前

如果以上情况均不符合，进⼊ key 的⽐较：

**oldKeyToIdx**：⼀个哈希表，存放旧节点的 key 与节点的映射关系；如果没有 oldKeyToIdx 则会通过 createKeyToOldIdx 会得到⼀个 oldKeyToIdx，⾥⾯存放旧节点的 key 与节点的映射关系，只不过这个 key 是 index 序列。从 oldKeyToIdx 这个哈希表中可以找到与新节点是否有相同 key的旧节点，如果同时满⾜ sameVnode，patchVnode 的同时会将这个真实 DOM（elmToMove）移动到 oldStartVnode 对应的真实 DOM 的前⾯。

**idxInOld**：拿新节点的 key 去 oldKeyToIdx 找是否有与旧节点相同的节点，即旧节点中是否有与新节点 key 相同的节点，没有就通过 ﬁndIdxInOld 遍历旧节点并通过 sameVnode 判断是否有相同节点，有返回索引。

idxInOld 不存在，即新节点在旧节点中都没有找到，说明这是⼀个之前没有的新节点，需要通过 createElm 创建新节点

idxInOld 存在，则进⼀步通过 sameVnode(vnodeToMove, newStartVnode) 判断是否是同

⼀节点

是同⼀节点，则通过 patchVnode 更新，并移动节点

不是同⼀节点，即相同的 key 不同的元素，则通过 createElm 创建新节点

先 与 两两通过 sameVnode 进⾏ 4

oldStartVnode、oldEndVnode

newStartVnode、newEndVnode

次⽐较，若成⽴，则通过 patchVnode 更新节点内容，并移动节点位置。若不成⽴，再进⼀步⽐较 key，idxInOld 判断新节点是否被旧节点复⽤了。idxInOld 不存在，说明旧节点没有复⽤新节点，新节点需要 createElm 创建；idxInOld 存在，说明新节点有被复⽤的可能性，为什么这么说，因为此时我们只知道节点的 key 相同，是否是通过简单的**通过移动节点位置达到复⽤的⽬的**，还是说通过创建节点进

⾏原地复⽤或就地修改，需要进⼀步通过 sameVnode(vnodeToMove, newStartVnode) 判断是否是同

⼀节点。是同⼀节点，则直接通过 patchVnode 更新，并移动节点；否则，虽然有相同的 key 但是不同

v-for

的元素，则通过 createElm 创建新节点，就地修改。从这⼀点我们就可以思考出为什么

要加上 key？为什么这个 key 不建议 index 来标识？

的时候

## Vue2 和 Vue3 响应性原理

Vue.js 2.0 采⽤数据劫持并结合了发布者-订阅者模式，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。

官⽹的描述：https://cn.vuejs.org/v2/guide/reactivity.html

Vue.js 3.0放弃了Object.deﬁneProperty API，⽽使⽤了更快的Proxy API。Proxy 是在 ES6 中引⼊，它允许你拦截对该对象的任何交互，也可以避免 Vue 早期版本中存在的⼀些响应性问题。

官⽹描述：https://v3.cn.vuejs.org/guide/reactivity.html

## nextTick ⽅法的实现原理

当调⽤nexttick函数时，nexttick内部会将回调函数使⽤Promise来包裹，⽬的是将该回调函数加⼊到微任务队列中。

在队列中的任务都是先进先出的，所以当执⾏完主程序的代码之后就会执⾏微任务队列中nexttick的回调函数，那这个过程就称为⼀次tick。

所以nexttick的回调函数将会推迟到下⼀个 DOM 更新周期之后执⾏。

## Proxy 与 Object.deﬁneProperty 优劣对⽐

### Proxy 的优势如下:

Object.deﬁneProperty 只能劫持对象的属性，⽽ Proxy 是直接代理对象。

由于 Object.deﬁneProperty 只能对属性进⾏劫持，需要遍历对象的每个属性，如果属性值也是对象，则需要深度遍历。⽽ Proxy 直接代理对象，不需要遍历操作。

Object.deﬁneProperty 对新增属性需要⼿动进⾏ Observe。

由于 Object.deﬁneProperty 劫持的是对象的属性，所以新增属性时，需要重新遍历对象，对其新增属性再使⽤ Object.deﬁneProperty 进⾏劫持。

也正是因为这个原因，使⽤ Vue 给 data 中的数组或对象新增属性时，需要使⽤ vm.$set 才能保证新增的属性也是响应式的。

Proxy⽀持 13 种拦截操作，这是 deﬁneProperty 所不具有的。

get(target, propKey, receiver)：拦截对象属性的读取，⽐如 proxy.foo 和proxy['foo']。 set(target, propKey, value, receiver)：拦截对象属性的设置，⽐如proxy.foo = v 或 proxy['foo'] = v，返回⼀个布尔值。

has(target, propKey)：拦截 propKey in proxy 的操作，返回⼀个布尔值。 deleteProperty(target, propKey)：拦截 delete proxy[propKey] 的操作，返回⼀个布尔值。 ownKeys(target)：拦截Object.getOwnPropertyNames(proxy)、 Object.getOwnPropertySymbols(proxy)、Object.keys(proxy)、for...in循环，返回⼀个数组。该⽅法返回⽬标对象所有⾃身的属性的属性名，⽽ Object.keys() 的返回结果仅包括⽬标对象⾃身的可遍历属性。

getOwnPropertyDescriptor(target, propKey)：拦截 Object.getOwnPropertyDescriptor(proxy, propKey)，返回属性的描述对象。 deﬁneProperty(target, propKey, propDesc)：拦截Object.deﬁneProperty(proxy, propKey, propDesc）、Object.deﬁneProperties(proxy, propDescs)，返回⼀个布尔值。 preventExtensions(target)：拦截 Object.preventExtensions(proxy)，返回⼀个布尔值。 getPrototypeOf(target)：拦截 Object.getPrototypeOf(proxy)，返回⼀个对象。 isExtensible(target)：拦截 Object.isExtensible(proxy)，返回⼀个布尔值。 setPrototypeOf(target, proto)：拦截 Object.setPrototypeOf(proxy, proto)，返回⼀个布尔值。如果⽬标对象是函数，那么还有两种额外操作可以拦截。

apply(target, object, args)：拦截 Proxy 实例作为函数调⽤的操作，⽐如proxy(...args)、

proxy.call(object, ...args)、proxy.apply(...)。

construct(target, args)：拦截 Proxy 实例作为构造函数调⽤的操作，⽐如new proxy(...args)。

新标准性能红利

Proxy 作为新标准，从⻓远来看，JS 引擎会继续优化 Proxy，但 getter 和 setter 基本不会再有针对性优化。

Proxy 兼容性差

Proxy 对于 IE 浏览器来说简直是灾难。 并且⽬前并没有⼀个完整⽀持 Proxy 所有拦截⽅法的 Polyﬁll ⽅案，有⼀个 Google 编写的 proxy-polyﬁll 也只⽀持了 get、set、apply、construct 四种拦截，可以⽀持到 IE9+ 和 Safari 6+。

### Object.deﬁneProperty 的优势如下:

兼容性好，⽀持 IE9。

### 最后总结：

* + 1. Object.deﬁneProperty 并⾮不能监控数组下标的变化，Vue2.x 中⽆法通过数组索引来实现响应式数据的⾃动更新是 Vue 本身的设计导致的，不是 deﬁneProperty 的锅。
    2. Object.deﬁneProperty 和 Proxy 本质差别是，deﬁneProperty 只能对属性进⾏劫持，所以出现了需要递归遍历，新增属性需要⼿动 Observe 的问题。
    3. Proxy 作为新标准，浏览器⼚商势必会对其进⾏持续优化，但它的兼容性也是块硬伤，并且⽬前还没有完整的 polyﬁll ⽅案。

## 使⽤过 Vue SSR 吗？说说 SSR？

Vue.js 是构建客户端应⽤程序的框架。默认情况下，可以在浏览器中输出 Vue 组件，进⾏⽣成 DOM 和操作 DOM。然⽽，也可以将同⼀个组件渲染为服务端的 HTML 字符串，将它们直接发送到浏览器，最后将这些静态标记"激活"为客户端上完全可交互的应⽤程序。

即：SSR⼤致的意思就是vue在客户端将标签渲染成的整个 html ⽚段的⼯作在服务端完成，服务端形成的html ⽚段直接返回给客户端这个过程就叫做服务端渲染。

### 服务端渲染 SSR 的优缺点如下：

**（1）服务端渲染的优点：**

更好的 SEO： 因为 SPA ⻚⾯的内容是通过 Ajax 获取，⽽搜索引擎爬取⼯具并不会等待 Ajax 异步完成后再抓取⻚⾯内容，所以在 SPA 中是抓取不到⻚⾯通过 Ajax 获取到的内容；⽽ SSR 是直接由服务端返回已经渲染好的⻚⾯（数据已经包含在⻚⾯中），所以搜索引擎爬取⼯具可以抓取渲染好的⻚⾯；

更快的内容到达时间（⾸屏加载更快）： SPA 会等待所有 Vue 编译后的 js ⽂件都下载完成后，才开始进⾏⻚⾯的渲染，⽂件下载等需要⼀定的时间等，所以⾸屏渲染需要⼀定的时间；SSR 直接由服务端渲染好⻚⾯直接返回显示，⽆需等待下载 js ⽂件及再去渲染等，所以 SSR 有更快的内容到达时间；

### （2) 服务端渲染的缺点：

更多的开发条件限制： 例如服务端渲染只⽀持 beforCreate 和 created 两个钩⼦函数，这会导致

⼀些外部扩展库需要特殊处理，才能在服务端渲染应⽤程序中运⾏；并且与可以部署在任何静态⽂件服务器上的完全静态单⻚⾯应⽤程序 SPA 不同，服务端渲染应⽤程序，需要处于 Node.js server 运⾏环境；

更多的服务器负载：在 Node.js 中渲染完整的应⽤程序，显然会⽐仅仅提供静态⽂件的 server 更加⼤量占⽤CPU 资源 (CPU-intensive - CPU 密集)，因此如果你预料在⾼流量环境 ( high traﬃc )下使⽤，请准备相应的服务器负载，并明智地采⽤缓存策略。

## 你有对 Vue 项⽬进⾏哪些优化？

### 代码层⾯的优化

v-if 和 v-show 区分使⽤场景

computed 和 watch 区分使⽤场景

v-for 遍历必须为 item 添加 key，且避免同时使⽤ v-if

⻓列表性能优化事件的销毁

图⽚资源懒加载路由懒加载

第三⽅插件的按需引⼊优化⽆限列表性能

服务端渲染 SSR or 预渲染

### 打包层⾯的优化

Webpack 对图⽚进⾏压缩 减少 ES6 转为 ES5 的冗余代码提取公共代码

模板预编译 提取组件的 CSS

优化 SourceMap

构建结果输出分析

Vue 项⽬的编译优化

### 其它的优化

开启 gzip 压缩浏览器缓存 CDN 的使⽤

使⽤ Chrome Performance 查找性能瓶颈

## 如何实现Vue⾸屏加载优化的

* + 1. 把不常改变的库放到index.html中，并接⼊CDN提速
    2. Vue路由的懒加载，Vue组件尽量不要全局引⼊
    3. 使⽤轻量级的⼯具库
    4. 减少⾸⻚的资源请求数，减少资源的⼤⼩
    5. 避免使⽤⼤图，图⽚使⽤懒加载
    6. 等等

## Vue3.0 ⾥为什么要⽤ Proxy API替代 deﬁneProperty API

deﬁneProperty API 的局限性最⼤原因是它只能针对对象的属性做监听。

Vue2.x中的响应式实现正是基于Object.deﬁneProperty来实现，对 data 中的属性做了遍历 + 递归，为每个属性设置了 getter、setter。这也就是为什么 Vue 只能对 data 中预定义过的属性做出响应的原因。

Proxy API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， 这就完全可以代理所有属性，将带来很⼤的性能提升和更优的代码。

Proxy 可以理解成，在⽬标对象之前架设⼀层“拦截”，外界对该对象的访问，都必须先通过这层拦截，因此提供了⼀种机制，可以对外界的访问进⾏过滤和改写。

Proxy响应式是惰性的。

在 Vue.js 2.x 中，对于⼀个深层属性嵌套的对象，要劫持它内部深层次的变化，就需要递归遍历这个对象，执⾏ Object.deﬁneProperty 把每⼀层对象数据都变成响应式的，这⽆疑会有很⼤的性能消耗。

在 Vue.js 3.0 中，使⽤ Proxy API 并不能监听到对象内部深层次的属性变化，因此它的处理⽅式是在 getter 中去递归响应式，这样的好处是真正访问到的内部属性才会变成响应式，简单的可以说是按需实现响应式，减少性能消耗。

# Axios⽹络请求库

## Axios是什么？怎么使⽤它，怎么解决跨域？

### Axios是什么？

Axios 是⼀个基于 promise 的 HTTP 库，可以⽤在浏览器和 node.js 中。前端最流⾏的 ajax 请求库，

react/vue 官⽅都推荐使⽤ axios 发 ajax 请求

### Axios特点:

基于 promise 的异步 ajax 请求库，⽀持promise所有的API

浏览器端/node 端都可以使⽤，浏览器中创建XMLHttpRequests

⽀持请求／响应拦截器

⽀持请求取消

JSON数据的⾃动转换

可以转换请求数据和响应数据，并对响应回来的内容⾃动转换成 JSON类型的数据批量发送多个请求

安全性更⾼，客户端⽀持防御 XSRF。

就是让你的每个请求都带⼀个从cookie中拿到的key, 根据浏览器同源策略，假冒的⽹站是拿不到你 cookie中得key的，这样，后台就可以轻松辨别出这个请求是否是⽤户在假冒⽹站上的误导输⼊，从⽽采取正确的策略。

### 常⽤语法：

axios(conﬁg): 通⽤/最本质的发任意类型请求的⽅式 axios.request(conﬁg): 等同于 axios(conﬁg) axios.get(url[, conﬁg]): 发 get 请求 axios.post(url[, data, conﬁg]): 发 post 请求 axios.defaults.xxx: 请求的默认全局配置 axios.interceptors.request.use(): 添加请求拦截器 axios.interceptors.response.use(): 添加响应拦截器

axios.create([conﬁg]): 创建⼀个新的 axios(它没有下⾯的功能) axios.CancelToken(): ⽤于创建取消请求的 token 对象

axios.all(promises): ⽤于批量执⾏多个异步请求

### 开发环境中解决跨域：

⽅式⼀：在vue.conﬁg.js中的devServer选项中的proxy中配置反向代理

⽅式⼆：在vite.conﬁg.js中的server选项中的proxy中配置反向代理

⽅式三：直接后端开发⼈员配置cors

### 部署的时候解决跨域：

⽅式⼀：直接后端开发⼈员配置cors

⽅式⼆：反向代理，⽐如：nginx 服务器的反向代理。

# Vue VS React

## Vue和React有什么不同？使⽤场景分别是什么？

### Vue和React的区别：

虽然Vue和React两者在定位上有⼀些交集，但差异也是很明显的。

### Vue 使⽤的是 web 开发者更熟悉的模板与特性。

Vue的API跟传统web开发者熟悉的模板契合度更⾼，⽐如Vue的单⽂件组件是以模板

+JavaScript+CSS的组合模式呈现，它跟web现有的HTML、JavaScript、CSS能够更好地配合。

### React 的特⾊在于函数式编程的理念和丰富的技术选型。

Vue ⽐起 React 更容易被前端⼯程师接受，这是⼀个直观的感受；React 则更容易吸引在 FP 上持续⾛下去的开发者。

从**使⽤习惯和思维模式**上考虑，对于⼀个没有任何Vue和React基础的web开发者来说， Vue会更友好，更符合他的思维模式。

React对于拥有函数式编程背景的开发者以及⼀些并不是以web为主要开发平台的开发⼈员⽽⾔， React更容易接受。

这并不意味着他们不能接受Vue，Vue和React之间的差异对他们来说就没有web开发者那么明显。可以说，**Vue更加注重web开发者的习惯**。

### 各⾃使⽤场景：

React：适合构建⼤型应⽤程序，其函数编程让React开发应⽤时⾮常的灵活，并且React对TS⽀持⾮常的友好，也有⾮常⼤的⽣态系统。

Vue：适合使⽤模板搭建⼩⽽快的应⽤；适合简单、对灵活度要求没那么⾼的应⽤。⾃从Vue3出来后，对TS⽀持也越来越友好，⽣态系统也在慢慢的完善起来了。