# 面试题锦集-vue

## vue项目，首次加载白屏时间较长的优化方式有哪些？

    1、减小项目体积html、css、js等  
    2、减少接口请求、静态资源、图片、字体包等  
    3、路由懒加载、组件懒加载  
    4、第三方库按需引入  
    5、第三方库通过cdn请求  
    6、减少需要加载/需要执行的脚本组织渲染。

## 计算属性computed和watch有什么区别?以及它们的运用场景?

// 区别

computed 计算属性：依赖其它属性值，并且computed的值有缓存，只有它依赖的属性值发生改变，下一次获取computed的值时才会重新计算computed的值。

watch 侦听器：更多的是观察的作用,无缓存性,类似与某些数据的监听回调,每当监听的数据变化时都会执行回调进行后续操作

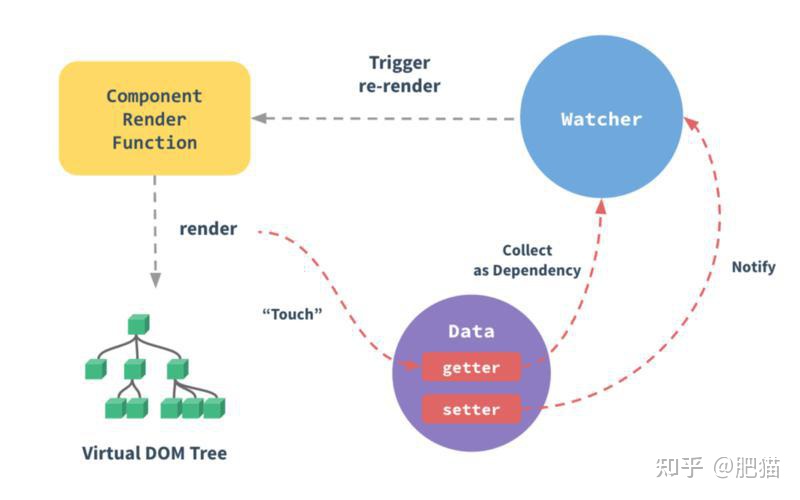
//运用场景

当需要进行数值计算,并且依赖与其它数据时,应该使用computed,因为可以利用computed的缓存属性,避免每次获取值时都要重新计算。

当需要在数据变化时执行异步或开销较大的操作时,应该使用watch,使用watch选项允许执行异步操作（访问一个API),限制执行该操作的频率，并在得到最终结果前，设置中间状态。这些都是计算属性无法做到的。

## 1. Vue 的基本原理

当一个Vue实例创建时，Vue会遍历data中的属性，用Object.defineProperty（vue3.0使用proxy）将它们转为getter/setter，并且在内部追踪相关依赖，在属性被访问和修改时通知变化。每个组件实例都有相应的watcher程序实例，它会在组件渲染的过程中把属性记录为依赖，之后当依赖项的setter被调用时，会通知watcher重新计算，从而使它关联的组件得以更新。



### 总结

1. 解析模版成render函数
2. 响应式开始监听
3. 首次渲染，显示页面且绑定依赖
4. data属性变化，触发rerender

Vue实现了数据和视图的分离，解耦，以数据驱动视图，只关系数据的变化，DOM操作被封装。

## 2. 双向数据绑定的原理

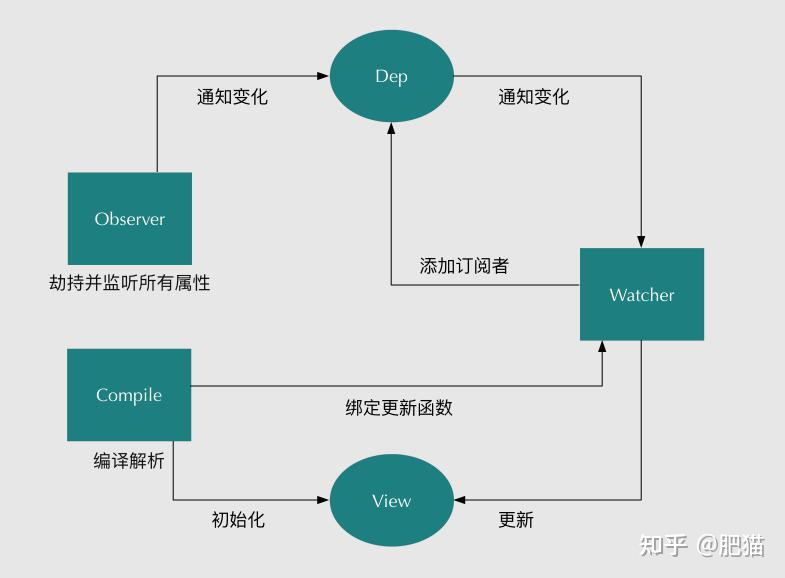
Vue.js 是采用数据劫持结合发布者-订阅者模式的方式，通过 Object.defineProperty()来劫持各个属性的 setter，getter，在数 据变动时发布消息给订阅者，触发相应的监听回调。主要分为以下几 个步骤：

1.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

2.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

3.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，则成功 成身退。

4.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。



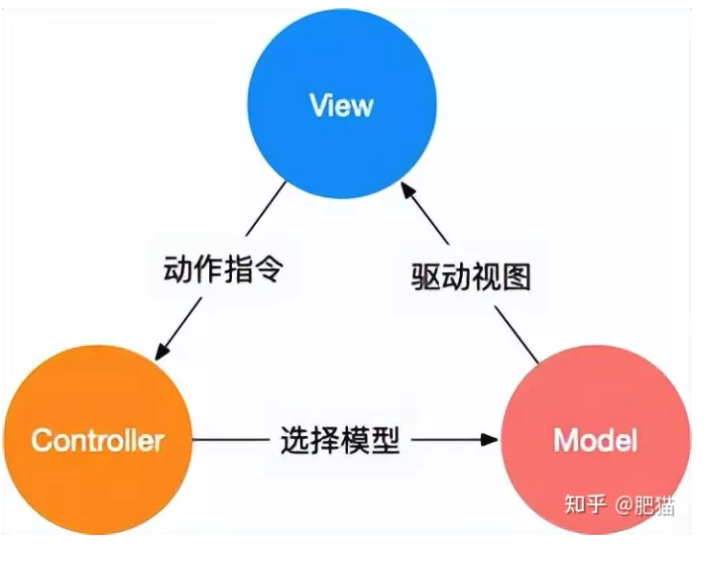
## 3. MVVM、MVC、MVP 的区别

MVC、MVP 和 MVVM 是三种常见的软件架构设计模式，主要通过分离 关注点的方式来组织代码结构，优化开发效率。

在开发单页面应用时，往往一个路由页面对应了一个脚本文件，所有 的页面逻辑都在一个脚本文件里。页面的渲染、数据的获取，对用户来说 事件的响应所有的应用逻辑都混合在一起，这样在开发简单项目时，可能看不出什么问题，如果项目变得复杂，那么整个文件就会变得冗 长、混乱，这样对项目开发和后期的项目维护是非常不利的。

### （1）MVC

MVC 通过分离 Model、View 和 Controller 的方式来组织代码结构。其中 View 负责页面的显示逻辑，Model 负责存储页面的业务数据，以及对相应数据的操作。并且 View 和 Model 应用了观察者模式，当 Model 层发生改变的时候它会通知有关 View 层更新页面。Controller 层是 View 层和 Model 层的纽带，它主要负责用户与应 用的响应操作，当用户与页面产生交互的时候，Controller 中的事 触发器就开始工作了，通过调用 Model 层，来完成对 Model 的修 改，然后 Model 层再去通知 View 层更新。



### （2）MVVM

MVVM 分为 Model、View、ViewModel：

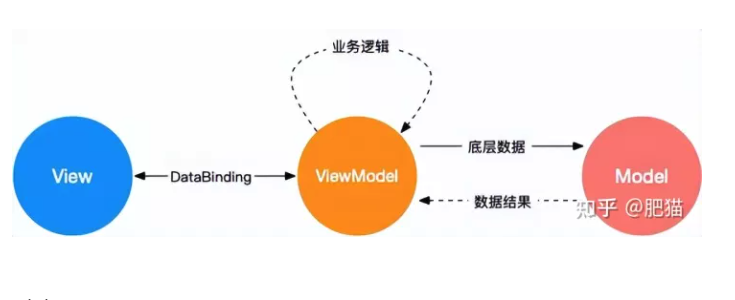
Model 代表数据模型，数据和业务逻辑都在 Model 层中定义；

View 代表 UI 视图，负责数据的展示；

ViewModel 负责监听 Model 中数据的改变并且控制视图的更新，处理 用户交互操作；

Model 和 View 并无直接关联，而是通过 ViewModel 来进行联系的，Model 和 ViewModel 之间有着双向数据绑定的联系。因此当 Model 中 的数据改变时会触发 View 层的刷新，View 中由于用户交互操作而改 变的数据也会在 Model 中同步。

这种模式实现了 Model 和 View 的数据自动同步，因此开发者只需要 专注于数据的维护操作即可，而不需要自己操作 DOM。



### （3）MVP

MVP 模式与 MVC 唯一不同的在于 Presenter 和 Controller。在 MVC 模式中使用观察者模式，来实现当 Model 层数据发生变化的时 候，通知 View 层的更新。这样 View 层和 Model 层耦合在一起，当项目逻辑变得复杂的时候，可能会造成代码的混乱，并且可能会对 代码的复用性造成一些问题。MVP 的模式通过使用 Presenter 来实 现对 View 层和 Model 层的解耦。MVC 中的 Controller 只知道 Model 的接口，因此它没有办法控制 View 层的更新，MVP 模式中，View 层的接口暴露给了 Presenter 因此可以在 Presenter 中将 Model 的变化和 View 的变化绑定在一起，以此来实现 View 和 Model 的同步更新。这样就实现了对 View 和 Model 的解耦，Presenter 还包含了其他的响应逻辑。

## 4. slot 是什么？有什么作用？原理是什么？

slot 又名插槽，是 Vue 的内容分发机制，组件内部的模板引擎使用 slot 元素作为承载分发内容的出口。插槽 slot 是子组件的一个模板 标签元素，而这一个标签元素是否显示，以及怎么显示是由父组件决 定的。slot 又分三类，默认插槽，具名插槽和作用域插槽。

默认插槽：又名匿名插槽，当 slot 没有指定 name 属性值的时候一个 默认显示插槽，一个组件内只有有一个匿名插槽。

具名插槽：带有具体名字的插槽，也就是带有 name 属性的 slot，一 个组件可以出现多个具名插槽。

作用域插槽：默认插槽、具名插槽的一个变体，可以是匿名插槽，也 可以是具名插槽，该插槽的不同点是在子组件渲染作用域插槽时，可 以将子组件内部的数据传递给父组件，让父组件根据子组件的传递过 来的数据决定如何渲染该插槽。

实现原理：当子组件 vm 实例化时，获取到父组件传入的 slot 标签的 内容，存放在 vm.$slot 中，默认插槽为 vm.$slot.default，具名插 槽为 vm.$[slot.xxx](https://link.zhihu.com/?target=http://slot.xxx" \t "/Users/gdj/Documents\\x/_blank)，xxx 为插槽名，当组件执行渲染函数时候，遇 到 slot 标签，使用$slot 中的内容进行替换，此时可以为插槽传递 数据，若存在数据，则可称该插槽为作用域插槽。

5. $nextTick 原理及作用

Vue 的 nextTick 其本质是对 JavaScript 执行原理 EventLoop 的 一种应用。

nextTick 的 核 心 是 利 用 了 如 Promise 、 MutationObserver 、setImmediate、setTimeout 的原生 JavaScript 方法来模拟对应的 微/宏任务的实现，本质是为了利用 JavaScript 的这些异步回调任 务队列来实现 Vue 框架中自己的异步回调队列。

nextTick 不仅是 Vue 内部的异步队列的调用方法，同时也允许开发 者在实际项目中使用这个方法来满足实际应用中对 DOM 更新数据时 机的后续逻辑处理

nextTick 是典型的将底层 JavaScript 执行原理应用到具体案例中 的示例，引入异步更新队列机制的原因∶如果是同步更新，则多次对一个或多个属性赋值，会频繁触发 UI/DOM 的渲染，可以减少一些无用渲染

同时由于 VirtualDOM 的引入，每一次状态发生变化后，状态变化的 信号会发送给组件，组件内部使用 VirtualDOM 进行计算得出需要更 新的具体的 DOM 节点，然后对 DOM 进行更新操作，每次更新状态后 的渲染过程需要更多的计算，而这种无用功也将浪费更多的性能，所 以异步渲染变得更加至关重要

Vue 采用了数据驱动视图的思想，但是在一些情况下，仍然需要操作 DOM。有时候，可能遇到这样的情况，DOM1 的数据发生了变化，而 DOM2 需要从 DOM1 中获取数据，那这时就会发现 DOM2 的视图并没有更新，这时就需要用到了 nextTick 了。

由于 Vue 的 DOM 操作是异步的，所以，在上面的情况中，就要将 DOM2 获取数据的操作写在$nextTick 中。



所以，在以下情况下，会用到 nextTick：

在数据变化后执行的某个操作，而这个操作需要使用随数据变化而变 化的 DOM 结构的时候，这个操作就需要方法在 nextTick()的回调函 数中。

在 vue 生命周期中，如果在 created()钩子进行 DOM 操作，也一定要 放在 nextTick()的回调函数中。

因为在 created()钩子函数中，页面的 DOM 还未渲染，这时候也没办 法操作 DOM，所以，此时如果想要操作 DOM，必须将操作的代码放在 nextTick()的回调函数中。

## 6. Vue 单页应用与多页应用的区别

概念：

SPA 单页面应用（SinglePage Web Application），指只有一个主页 面的应用，一开始只需要加载一次 js、css 等相关资源。所有内容都 包含在主页面，对每一个功能模块组件化。单页应用跳转，就是切换 相关组件，仅仅刷新局部资源。

MPA 多页面应用 （MultiPage Application），指有多个独立页面的 应用，每个页面必须重复加载 js、css 等相关资源。多页应用跳转，需要整页资源刷新。

## 7. Vue 中封装的数组方法有哪些，其如何实现页面更新

在 Vue 中，对响应式处理利用的是 Object.defineProperty 对数据进行拦截，而这个方法并不能监听到数组内部变化，数组长度变化，数组的截取变化等，所以需要对这些操作进行 hack，让 Vue 能监听到 其中的变化。

简单来说就是，重写了数组中的那些原生方法，首先获取到这个数组 的\_\_ob\_\_，也就是它的 Observer 对象，如果有新的值，就调用 observeArray 继续对新的值观察变化（也就是通过 target\_\_proto\_\_ == arrayMethods 来改变了数组实例的型），然后手动调用 notify，通知渲染 watcher，执行 update。

## 8. 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。

## 9. 简述 mixin、extends 的覆盖逻辑

### （1）mixin 和 extends

mixin 和 extends 均 是 用 于 合 并 、 拓 展 组 件 的 ， 两 者 均 通 过 mergeOptions 方法实现合并。

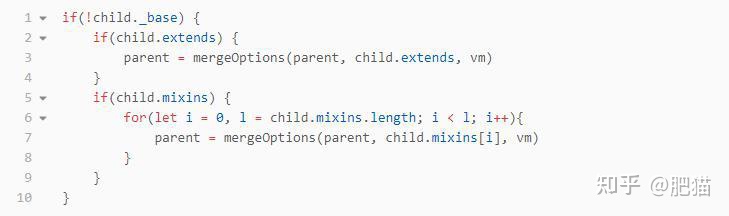
mixins 接收一个混入对象的数组，其中混入对象可以像正常的实例 对象一样包含实例选项，这些选项会被合并到最终的选项中。Mixin 钩子按照传入顺序依次调用，并在调用组件自身的钩子之前被调用。

extends 主要是为了便于扩展单文件组件，接收一个对象或构造函数。

### （2）mergeOptions 的执行过程

规 范 化 选 项 （ normalizeProps 、 normalizelnject 、normalizeDirectives)

对未合并的选项，进行判断



## 10. 子组件可以直接改变父组件的数据吗？

子组件不可以直接改变父组件的数据。这样做主要是为了维护父子组 件的单向数据流。每次父级组件发生更新时，子组件中所有的 prop 都将会刷新为最新的值。如果这样做了，Vue 会在浏览器的控制台中 发出警告。Vue 提倡单向数据流，即父级 props 的更新会流向子组件，但是反 过来则不行。这是为了防止意外的改变父组件状态，使得应用的数据 流变得难以理解，导致数据流混乱。如果破坏了单向数据流，当应用 复杂时，debug 的成本会非常高。

只能通过 $emit 派发一个自定义事件，父组件接收到后，由父组件 修改。

## 11. 对 React 和 Vue 的理解，它们的异同

相似之处：

都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；

都有自己的构建工具，能让你得到一个根据最佳实践设置的项目模板；都使用了 Virtual DOM（虚拟 DOM）提高重绘性能；

都有 props 的概念，允许组件间的数据传递；

都鼓励组件化应用，将应用分拆成一个个功能明确的模块，提高复用 性。

不同之处 ：

1）数据流

Vue 默认支持数据双向绑定，而 React 一直提倡单向数据流

2）虚拟 DOM

Vue2.x 开始引入"Virtual DOM"，消除了和 React 在这方面的差异，但是在具体的细节还是有各自的特点。Vue 宣称可以更快地计算出 Virtual DOM 的差异，这是由于它在渲染 过程中，会跟踪每一个组件的依赖关系，不需要重新渲染整个组件树。

对于 React 而言，每当应用的状态被改变时，全部子组件都会重新渲 染。当然，这可以通过 PureComponent/shouldComponentUpdate 这 个生命周期方法来进行控制，但 Vue 将此视为默认的优化。

3）组件化

React 与 Vue 最大的不同是模板的编写。

Vue 鼓励写近似常规 HTML 的模板。写起来很接近标准 HTML 元素，只 是多了一些属性。

React 推荐你所有的模板通用 JavaScript 的语法扩展——JSX 书写。

具体来讲：React 中 render 函数是支持闭包特性的，所以 import 的 组件在 render 中可以直接调用。但是在 Vue 中，由于模板中使用的 数据都必须挂在 this 上进行一次中转，所以 import 一个组件完了 之后，还需要在 components 中再声明下。

4）监听数据变化的实现原理不同

Vue 通过 getter/setter 以及一些函数的劫持，能精确知道数据变 化，不需要特别的优化就能达到很好的性能

React 默 认 是 通 过 比 较 引 用 的 方 式 进 行 的 ， 如 果 不 优 化（PureComponent/shouldComponentUpdate）可能导致大量不必要的 vDOM 的重新渲染。这是因为 Vue 使用的是可变数据，而 React 更强 调数据的不可变。

5）高阶组件

react 可以通过高阶组件（HOC）来扩展，而 Vue 需要通过 mixins 来 扩展。

高阶组件就是高阶函数，而 React 的组件本身就是纯粹的函数，所以 高阶函数对 React 来说易如反掌。相反 Vue.js 使用 HTML 模板创建视 图组件，这时模板无法有效的编译，因此 Vue 不能采用 HOC 来实现。

6）构建工具

两者都有自己的构建工具：

React ==> Create React APP

Vue ==> vue-cli

7）跨平台

React ==> React Native

Vue ==> Weex

## 12. Vue 的优点

轻量级框架：只关注视图层，是一个构建数据的视图集合，大小只有 几十 kb ；

简单易学：国人开发，中文文档，不存在语言障碍 ，易于理解和学 习；

双向数据绑定：保留了 angular 的特点，在数据操作方面更为简单；组件化：保留了 react 的优点，实现了 html 的封装和重用，在构 建单页面应用方面有着独特的优势；

视图，数据，结构分离：使数据的更改更为简单，不需要进行逻辑代 码的修改，只需要操作数据就能完成相关操作；

虚拟 DOM：dom 操作是非常耗费性能的，不再使用原生的 dom 操作 节点，极大解放 dom 操作，但具体操作的还是 dom 不过是换了另一 种方式；

运行速度更快：相比较于 react 而言，同样是操作虚拟 dom，就性 能而言， vue 存在很大的优势。

## 13. assets 和 static 的区别

相同点： assets 和 static 两个都是存放静态资源文件。项目中所 需要的资源文件图片，字体图标，样式文件等都可以放在这两个文件 下，这是相同点

不相同点：assets 中存放的静态资源文件在项目打包时，也就是运 行 npm run build 时会将 assets 中放置的静态资源文件进行打包 上传，所谓打包简单点可以理解为压缩体积，代码格式化。而压缩后 的静态资源文件最终也都会放置在 static 文件中跟着 index.html 一同上传至服务器。static 中放置的静态资源文件就不会要走打包 压缩格式化等流程，而是直接进入打包好的目录，直接上传至服务器。

因为避免了压缩直接进行上传，在打包时会提高一定的效率，但是 static 中的资源文件由于没有进行压缩等操作，所以文件的体积也 就相对于 assets 中打包后的文件提交较大点。在服务器中就会占据 更大的空间。

建议： 将项目中 template 需要的样式文件 js 文件等都可以放置在 assets 中，走打包这一流程。减少体积。而项目中引入的第三方的 资源文件如 iconfoont.css 等文件可以放置在 static 中，因为这 些引入的第三方文件已经经过处理，不再需要处理，直接上传。

## 14. delete 和 Vue.delete 删除数组的区别

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。

## 15. Vue 模版编译原理

vue 中的模板 template 无法被浏览器解析并渲染，因为这不属于浏 览器的标准，不是正确的 HTML 语法，所有需要将 template 转化成一 个 JavaScript 函数，这样浏览器就可以执行这一个函数并渲染出对 应的 HTML 元素，就可以让视图跑起来了，这一个转化的过程，就成 为模板编译。模板编译又分三个阶段，解析 parse，优化 optimize，生成 generate，最终生成可执行函数 render。

解析阶段：使用大量的正则表达式对 template 字符串进行解析，将 标签、指令、属性等转化为抽象语法树 AST。

优化阶段：遍历 AST，找到其中的一些静态节点并进行标记，方便在 页面重渲染的时候进行 diff 比较时，直接跳过这一些静态节点，优 化 runtime 的性能。

生成阶段：将最终的 AST 转化为 render 函数字符串。

## 16. vue 初始化页面闪动问题

使用 vue 开发时，在 vue 初始化之前，由于 div 是不归 vue 管的，所 以我们写的代码在还没有解析的情况下会容易出现花屏现象，看到类 似于{{message}}的字样，虽然一般情况下这个时间很短暂，但是还 是有必要让解决这个问题的。

首先：在 css 里加上以下代码：



如 果 没 有 彻 底 解 决 问 题 ， 则 在 根 元 素 加 上 style="display:

none;" :style="{display: 'block'}"

## 17. MVVM 的优缺点?

优点:

分离视图（View）和模型（Model），降低代码耦合，提⾼视图或者 逻辑的重⽤性: ⽐如视图（View）可以独⽴于 Model 变化和修改，⼀个 ViewModel 可以绑定不同的"View"上，当 View 变化的时候 Model 不可以不变，当 Model 变化的时候 View 也可以不变。你可以把⼀些 视图逻辑放在⼀个 ViewModel⾥⾯，让很多 view 重⽤这段视图逻辑

提⾼可测试性: ViewModel 的存在可以帮助开发者更好地编写测试代 码

⾃动更新 dom: 利⽤双向绑定,数据更新后视图⾃动更新,让开发者从 繁琐的⼿动 dom 中解放

缺点:

Bug 很难被调试: 因为使⽤双向绑定的模式，当你看到界⾯异常了，有可能是你 View 的代码有 Bug，也可能是 Model 的代码有问题。数 据绑定使得⼀个位置的 Bug 被快速传递到别的位置，要定位原始出问 题的地⽅就变得不那么容易了。另外，数据绑定的声明是指令式地写 在 View 的模版当中的，这些内容是没办法去打断点 debug 的

⼀个⼤的模块中 model 也会很⼤，虽然使⽤⽅便了也很容易保证了数 据的⼀致性，当时⻓期持有，不释放内存就造成了花费更多的内存

对于⼤型的图形应⽤程序，视图状态较多，ViewModel 的构建和维护 的成本都会⽐较⾼。

18. v-if 和 v-for 哪个优先级更高？如果同时出现，应如何优 化？

v-for 优先于 v-if 被解析，如果同时出现，每次渲染都会先执行循 环再判断条件，无论如何循环都不可避免，浪费了性能。

要避免出现这种情况，则在外层嵌套 template，在这一层进行 v-if 判断，然后在内部进行 v-for 循环。如果条件出现在循环内部，可通 过计算属性提前过滤掉那些不需要显示的项。

## 19. 对 Vue 组件化的理解

1.组件是独立和可复用的代码组织单元。组件系统是 Vue 核心特性之 一，它使开发者使用小型、独立和通常可复用的组件构建大型应用；

2.组件化开发能大幅提高应用开发效率、测试性、复用性等；

3.组件使用按分类有：页面组件、业务组件、通用组件；

4.vue 的组件是基于配置的，我们通常编写的组件是组件配置而非组 件，框架后续会生成其构造函数，它们基于 VueComponent，扩展于 Vue；

5.vue 中常见组件化技术有：属性 prop，自定义事件，插槽等，它们 主要用于组件通信、扩展等；6.合理的划分组件，有助于提升应用性 能；

6.组件应该是高内聚、低耦合的；

7.遵循单向数据流的原则。

## 20. 对 vue 设计原则的理解

1.渐进式 JavaScript 框架：与其它大型框架不同的是，Vue 被设计 为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上 手，还便于与第三方库或既有项目整合。另一方面，当与现代化的工 具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应 用提供驱动。

2.易用性：vue 提供数据响应式、声明式模板语法和基于配置的组件 系统等核心特性。这些使我们只需要关注应用的核心业务即可，只要 会写 js、html 和 css 就能轻松编写 vue 应用。

3.灵活性：渐进式框架的最大优点就是灵活性，如果应用足够小，我 们可能仅需要 vue 核心特性即可完成功能；随着应用规模不断扩大，我们才可能逐渐引入路由、状态管理、vue-cli 等库和工具，不管是 应用体积还是学习难度都是一个逐渐增加的平和曲线。

4.高效性：超快的虚拟 DOM 和 diff算法使我们的应用拥有最佳的性能 表现。追求高效的过程还在继续，vue3 中引入 Proxy 对数据响应式 改进以及编译器中对于静态内容编译的改进都会让 vue 更加高效。

## 21. 说一下 Vue 的生命周期

Vue 实例有⼀个完整的⽣命周期，也就是从开始创建、初始化数据、编译模版、挂载 Dom -> 渲染、更新 -> 渲染、卸载 等⼀系列过程，称这是 Vue 的⽣命周期。

1.beforeCreate（创建前）：数据观测和初始化事件还未开始，此时 data 的响应式追踪、event/watcher 都还没有被设置，也就是说不 能访问到 data、computed、watch、methods 上的方法和数据。

2.created（创建后） ：实例创建完成，实例上配置的 options 包 括 data、computed、watch、methods 等都配置完成，但是此时渲染 得节点还未挂载到 DOM，所以不能访问到 $el 属性。

3.beforeMount（挂载前）：在挂载开始之前被调用，相关的 render 函数首次被调用。实例已完成以下的配置：编译模板，把 data 里面 的数据和模板生成 html。此时还没有挂载 html 到页面上。

4.mounted（挂载后）：在 el 被新创建的 vm.$el 替换，并挂载到实 例上去之后调用。实例已完成以下的配置：用上面编译好的 html 内 容替换 el 属性指向的 DOM 对象。完成模板中的 html 渲染到 html 页 面中。此过程中进行 ajax 交互。

5.beforeUpdate（更新前）：响应式数据更新时调用，此时虽然响应 式数据更新了，但是对应的真实 DOM 还没有被渲染。

6.updated（更新后） ：在由于数据更改导致的虚拟 DOM 重新渲染和 打补丁之后调用。此时 DOM 已经根据响应式数据的变化更新了。调 用时，组件 DOM 已经更新，所以可以执行依赖于 DOM 的操作。然而 在大多数情况下，应该避免在此期间更改状态，因为这可能会导致更 新无限循环。该钩子在服务器端渲染期间不被调用。

7.beforeDestroy（销毁前）：实例销毁之前调用。这一步，实例仍 然完全可用，this 仍能获取到实例。

8.destroyed（销毁后）：实例销毁后调用，调用后，Vue 实例指示 的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例 也会被销毁。该钩子在服务端渲染期间不被调用。

另外还有 keep-alive 独有的生命周期，分别为 activated 和

deactivated。用 keep-alive 包裹的组件在切换时不会进行销毁，而 是缓存到内存中并执行 deactivated 钩子函数，命中缓存渲染后会执 行 activated 钩子函数。

## 22. Vue 子组件和父组件执行顺序 加载渲染过程：

1.父组件 beforeCreate

2.父组件 created

3.父组件 beforeMount

4.子组件 beforeCreate

5.子组件 created

6.子组件 beforeMount

7.子组件 mounted

8.父组件 mounted

更新过程：

1. 父组件 beforeUpdate

2.子组件 beforeUpdate

3.子组件 updated

4.父组件 updated

销毁过程：

1. 父组件 beforeDestroy

2.子组件 beforeDestroy

3.子组件 destroyed

4.父组件 destoryed

## 23. created 和 mounted 的区别

created:在模板渲染成 html 前调用，即通常初始化某些属性值，然 后再渲染成视图。

mounted:在模板渲染成 html 后调用，通常是初始化页面完成后，再 对 html 的 dom 节点进行一些需要的操作。

## 一般在哪个生命周期请求异步数据

我们可以在钩子函数 created、beforeMount、mounted 中进行调用，因为在这三个钩子函数中，data 已经创建，可以将服务端端返回的 数据进行赋值。

推荐在 created 钩子函数中调用异步请求，因为在 created 钩子函 数中调用异步请求有以下优点：

能更快获取到服务端数据，减少页面加载时间，用户体验更好；

SSR 不支持 beforeMount 、mounted 钩子函数，放在 created 中有 助于一致性。

## 24. keep-alive 中的生命周期哪些

keep-alive 是 Vue 提供的一个内置组件，用来对组件进行缓存——在组件切换过程中将状态保留在内存中，防止重复渲染 DOM。

如果为一个组件包裹了 keep-alive，那么它会多出两个生命周期：deactivated、activated。同时，beforeDestroy 和 destroyed 就 不会再被触发了，因为组件不会被真正销毁。

当组件被换掉时，会被缓存到内存中、触发 deactivated 生命周期；当组件被切回来时，再去缓存里找这个组件、触发 activated 钩子 函数。

## 25. 路由的 hash 和 history 模式的区别

Vue-Router 有两种模式：hash 模式和 history 模式。默认的路由模 式是 hash 模式。

1. hash 模式

简介： hash 模式是开发中默认的模式，它的 URL 带着一个#，例如：，它的 hash 值就是#/vue。

特点：hash 值会出现在 URL 里面，但是不会出现在 HTTP 请求中，对 后端完全没有影响。所以改变 hash 值，不会重新加载页面。这种模 式的浏览器支持度很好，低版本的 IE 浏览器也支持这种模式。hash 路由被称为是前端路由，已经成为 SPA（单页面应用）的标配。

原理： hash 模式的主要原理就是 onhashchange()事件：



使用 onhashchange()事件的好处就是，在页面的 hash 值发生变化时，无需向后端发起请求，window 就可以监听事件的改变，并按规则加 载相应的代码。除此之外，hash 值变化对应的 URL 都会被浏览器记

录下来，这样浏览器就能实现页面的前进和后退。虽然是没有请求后 端服务器，但是页面的 hash 值和对应的 URL 关联起来了。

2. history 模式

简介： history 模式的 URL 中没有#，它使用的是传统的路由分发模 式，即用户在输入一个 URL 时，服务器会接收这个请求，并解析这个 URL，然后做出相应的逻辑处理。

特 点 ：当 使 用 history 模 式 时 ， URL 就 像 这 样 ：。相比 hash 模式更加好看。但是，history 模式需要后台配置支持。如果后台没有正确配置，访问时会返回 404。

API： history api 可以分为两大部分，切换历史状态和修改历史状 态：

修 改 历 史 状 态 ： 包 括 了 HTML5 History Interface 中 新 增 的 pushState() 和 replaceState() 方法，这两个方法应用于浏览器的 历史记录栈，提供了对历史记录进行修改的功能。只是当他们进行修 改时，虽然修改了 url，但浏览器不会立即向后端发送请求。如果要 做到改变 url 但又不刷新页面的效果，就需要前端用上这两个 API。

切换历史状态： 包括 forward()、back()、go()三个方法，对应浏 览器的前进，后退，跳转操作。

虽然 history 模式丢弃了丑陋的#。但是，它也有自己的缺点，就是 在刷新页面的时候，如果没有相应的路由或资源，就会刷出 404 来。

如果想要切换到 history 模式，就要进行以下配置（后端也要进行配 置）：



3. 两种模式对比

调用 history.pushState() 相比于直接修改 hash，存在以下优势:

pushState() 设置的新 URL 可以是与当前 URL 同源的任意 URL；而 hash 只可修改 # 后面的部分，因此只能设置与当前 URL 同文档的 URL；

pushState() 设置的新 URL 可以与当前 URL 一模一样，这样也会把 记录添加到栈中；而 hash 设置的新值必须与原来不一样才会触发动 作将记录添加到栈中；

pushState() 通过 stateObject 参数可以添加任意类型的数据到记 录中；而 hash 只可添加短字符串；

pushState() 可额外设置 title 属性供后续使用。

hash 模式下，仅 hash 符号之前的 url 会被包含在请求中，后端如果 没有做到对路由的全覆盖，也不会返回 404 错误；history 模式下，前端的 url 必须和实际向后端发起请求的 url 一致，如果没有对用的 路由处理，将返回 404 错误。

hash 模式和 history 模式都有各自的优势和缺陷，还是要根据实际 情况选择性的使用。

## 26. Vue-router 跳转和 location.href 有什么区别

使用 location.href= /url 来跳转，简单方便，但是刷新了页面；

使用 history.pushState( /url ) ，无刷新页面，静态跳转；

引进 router ，然后使用 router.push( /url ) 来跳转，使用了 diff 算法，实现了按需加载，减少了 dom 的消耗。其实使用 router 跳 转和使用 history.pushState() 没什么差别的，因为 vue-router 就 是用了 history.pushState() ，尤其是在 history 模式下。

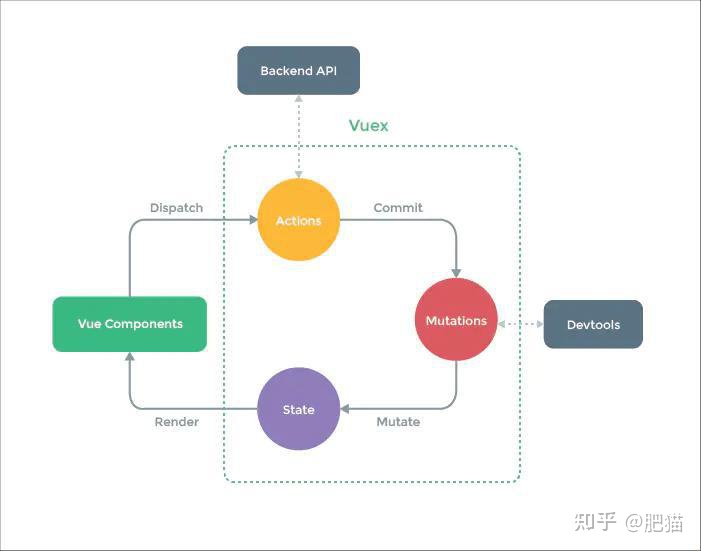
## 27. Vuex 的原理

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。这样可以方便地跟踪每一个状态的变化。



Vuex 为 Vue Components 建立起了一个完整的生态圈，包括开发中的 API 调用一环。

（1）核心流程中的主要功能：

Vue Components 是 vue 组件，组件会触发（dispatch）一些事件或 动作，也就是图中的 Actions;

在组件中发出的动作，肯定是想获取或者改变数据的，但是在 vuex 中，数据是集中管理的，不能直接去更改数据，所以会把这个动作提 交（Commit）到 Mutations 中;

然后 Mutations 就去改变（Mutate）State 中的数据;

当 State 中的数据被改变之后，就会重新渲染（Render）到 Vue Components 中去，组件展示更新后的数据，完成一个流程。

（2）各模块在核心流程中的主要功能：

Vue Components∶ Vue 组件。HTML 页面上，负责接收用户操作等交 互行为，执行 dispatch 方法触发对应 action 进行回应。

dispatch∶操作行为触发方法，是唯一能执行 action 的方法。

actions∶ 操作行为处理模块。负责处理 Vue Components 接收到的 所有交互行为。包含同步/异步操作，支持多个同名方法，按照注册 的顺序依次触发。向后台 API 请求的操作就在这个模块中进行，包括 触发其他 action 以及提交 mutation 的操作。该模块提供了 Promise 的封装，以支持 action 的链式触发。

commit∶状态改变提交操作方法。对 mutation 进行提交，是唯一能 执行 mutation 的方法。

mutations∶状态改变操作方法。是 Vuex 修改 state 的唯一推荐方法，其他修改方式在严格模式下将会报错。该方法只能进行同步操作，且 方法名只能全局唯一。操作之中会有一些 hook 暴露出来，以进行 state 的监控等。

state∶ 页面状态管理容器对象。集中存储 Vuecomponents 中 data 对象的零散数据，全局唯一，以进行统一的状态管理。页面显示所需 的数据从该对象中进行读取，利用 Vue 的细粒度数据响应机制来进行 高效的状态更新。

getters∶ state 对象读取方法。图中没有单独列出该模块，应该被 包含在了 render 中，Vue Components 通过该方法读取全局 state 对 象。

总结：

Vuex 实现了一个单向数据流，在全局拥有一个 State 存放数据，当 组件要更改 State 中的数据时，必须通过 Mutation 提交修改信息，Mutation 同时提供了订阅者模式供外部插件调用获取 State 数据 的更新。而当所有异步操作(常见于调用后端接口异步获取更新数据) 或批量的同步操作需要走 Action ，但 Action 也是无法直接修改 State 的，还是需要通过 Mutation 来修改 State 的数据。最后，根 据 State 的变化，渲染到视图上。

## 28. Vuex 和 localStorage 的区别

（1）最重要的区别

vuex 存储在内存中

localstorage 则以文件的方式存储在本地，只能存储字符串类型的 数据，存储对象需要 JSON 的 stringify 和 parse 方法进行处理。 读 取内存比读取硬盘速度要快

（2）应用场景

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。它采用集 中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一 种可预测的方式发生变化。vuex 用于组件之间的传值。

localstorage 是本地存储，是将数据存储到浏览器的方法，一般是 在跨页面传递数据时使用 。

Vuex 能做到数据的响应式，localstorage 不能

（3）永久性

刷新页面时 vuex 存储的值会丢失，localstorage 不会。

注意：对于不变的数据确实可以用 localstorage 可以代替 vuex，但 是当两个组件共用一个数据源（对象或数组）时，如果其中一个组件 改变了该数据源，希望另一个组件响应该变化时，localstorage 无 法做到，原因就是区别 1。

## 29. Redux 和 Vuex 有什么区别，它们的共同思想

（1）Redux 和 Vuex 区别

Vuex 改进了 Redux 中的 Action 和 Reducer 函数，以 mutations 变化 函数取代 Reducer，无需 switch，只需在对应的 mutation 函数里改 变 state 值即可

Vuex 由于 Vue 自动重新渲染的特性，无需订阅重新渲染函数，只要 生成新的 State 即可

Vuex 数据流的顺序是∶View 调用 store.commit 提交对应的请求到 Store 中对应的 mutation 函数->store 改变（vue 检测到数据变化自 动渲染）

通俗点理解就是，vuex 弱化 dispatch，通过 commit 进行 store 状 态的一次更变;取消了 action 概念，不必传入特定的 action 形式进 行指定变更;弱化 reducer，基于 commit 参数直接对数据进行转变，使得框架更加简易;

（2）共同思想

单—的数据源

变化可以预测

本质上：redux 与 vuex 都是对 mvvm 思想的服务，将数据从视图中抽 离的一种方案;

形式上：vuex 借鉴了 redux，将 store 作为全局的数据中心，进行 mode 管理;

## 30. 为什么要用 Vuex 或者 Redux

由于传参的方法对于多层嵌套的组件将会非常繁琐，并且对于兄弟组 件间的状态传递无能为力。我们经常会采用父子组件直接引用或者通 过事件来变更和同步状态的多份拷贝。以上的这些模式非常脆弱，通 常会导致代码无法维护。

所以需要把组件的共享状态抽取出来，以一个全局单例模式管理。在 这种模式下，组件树构成了一个巨大的"视图"，不管在树的哪个位置，任何组件都能获取状态或者触发行为。

另外，通过定义和隔离状态管理中的各种概念并强制遵守一定的规则，代码将会变得更结构化且易维护。

## 31. Vuex 有哪几种属性？

有五种，分别是 State、 Getter、Mutation 、Action、 Module

state => 基本数据(数据源存放地)

getters => 从基本数据派生出来的数据

mutations => 提交更改数据的方法，同步

actions => 像一个装饰器，包裹 mutations，使之可以异步。

modules => 模块化 Vuex

## 32. Vuex 和单纯的全局对象有什么区别？

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。

不能直接改变 store 中的状态。改变 store 中的状态的唯一途径就 是显式地提交 (commit) mutation。这样可以方便地跟踪每一个状态 的变化，从而能够实现一些工具帮助更好地了解我们的应用。

## 33. 为什么 Vuex 的 mutation 中不能做异步操作？

Vuex 中所有的状态更新的唯一途径都是 mutation，异步操作通过 Action 来提交 mutation 实现，这样可以方便地跟踪每一个状态的 变化，从而能够实现一些工具帮助更好地了解我们的应用。

每个 mutation 执行完成后都会对应到一个新的状态变更，这样 devtools 就可以打个快照存下来，然后就可以实现 time-travel 了。

如果 mutation 支持异步操作，就没有办法知道状态是何时更新的，无法很好的进行状态的追踪，给调试带来困难。

## 34. Vue3.0 有什么更新

（1）监测机制的改变

3.0 将带来基于代理 Proxy 的 observer 实现，提供全语言覆盖的 反应性跟踪。

消除了 Vue 2 当中基于 Object.defineProperty 的实现所存在的 很多限制：

（2）只能监测属性，不能监测对象

检测属性的添加和删除；

检测数组索引和长度的变更；

支持 Map、Set、WeakMap 和 WeakSet。

（3）模板

作用域插槽，2.x 的机制导致作用域插槽变了，父组件会重新渲染，而 3.0 把作用域插槽改成了函数的方式，这样只会影响子组件的重 新渲染，提升了渲染的性能。

同时，对于 render 函数的方面，vue3.0 也会进行一系列更改来方 便习惯直接使用 api 来生成 vdom 。

（4）对象式的组件声明方式

vue2.x 中 的 组 件 是 通 过 声 明 的 方 式 传 入 一 系 列 option ， 和 TypeScript 的结合需要通过一些装饰器的方式来做，虽然能实现功 能，但是比较麻烦。

3.0 修改了组件的声明方式，改成了类式的写法，这样使得和 TypeScript 的结合变得很容易

（5）其它方面的更改

支持自定义渲染器，从而使得 weex 可以通过自定义渲染器的方式来 扩展，而不是直接 fork 源码来改的方式。

支持 Fragment（多个根节点）和 Protal（在 dom 其他部分渲染组 建内容）组件，针对一些特殊的场景做了处理。

基于 tree shaking 优化，提供了更多的内置功能。

## 35. defineProperty 和 proxy 的区别

Vue 在 实 例 初 始 化 时 遍 历 data 中 的 所 有 属 性 ， 并 使 用 Object.defineProperty 把这些属性全部转为 getter/setter。这样 当追踪数据发生变化时，setter 会被自动调用。

Object.defineProperty 是 ES5 中一个无法 shim 的特性，这也就 是 Vue 不支持 IE8 以及更低版本浏览器的原因。

但是这样做有以下问题：

1.添加或删除对象的属性时，Vue 检测不到。因为添加或删除的对象 没 有 在 初 始 化 进 行 响 应 式 处 理 ， 只 能 通 过 $set 来 调 用 Object.defineProperty()处理。

2.无法监控到数组下标和长度的变化。

Vue3 使用 Proxy 来监控数据的变化。Proxy 是 ES6 中提供的功能，其作用为：用于定义基本操作的自定义行为（如属性查找，赋值，枚 举，函数调用等）。相对于 Object.defineProperty()，其有以下特 点：

1.Proxy 直接代理整个对象而非对象属性，这样只需做一层代理就可 以监听同级结构下的所有属性变化，包括新增属性和删除属性。2.Proxy 可以监听数组的变化。

## 36. Vue3.0 为什么要用 proxy？

在 Vue2 中， 0bject.defineProperty 会改变原始数据，而 Proxy 是创建对象的虚拟表示，并提供 set 、get 和 deleteProperty 等 处理器，这些处理器可在访问或修改原始对象上的属性时进行拦截，有以下特点∶

不需用使用 Vue.$set 或 Vue.$delete 触发响应式。

全方位的数组变化检测，消除了 Vue2 无效的边界情况。

支持 Map，Set，WeakMap 和 WeakSet。

Proxy 实现的响应式原理与 Vue2 的实现原理相同，实现方式大同小 异∶

get 收集依赖

Set、delete 等触发依赖

对于集合类型，就是对集合对象的方法做一层包装：原方法执行后执 行依赖相关的收集或触发逻辑。

## 37. 虚拟 DOM 的解析过程

虚拟 DOM 的解析过程：

首先对将要插入到文档中的 DOM 树结构进行分析，使用 js 对象将 其表示出来，比如一个元素对象，包含 TagName、props 和 Children

这些属性。然后将这个 js 对象树给保存下来，最后再将 DOM 片段 插入到文档中。

当页面的状态发生改变，需要对页面的 DOM 的结构进行调整的时候，首先根据变更的状态，重新构建起一棵对象树，然后将这棵新的对象 树和旧的对象树进行比较，记录下两棵树的的差异。

最后将记录的有差异的地方应用到真正的 DOM 树中去，这样视图就 更新了。

## 38. DIFF 算法的原理

在新老虚拟 DOM 对比时：

首先，对比节点本身，判断是否为同一节点，如果不为相同节点，则 删除该节点重新创建节点进行替换

如果为相同节点，进行 patchVnode，判断如何对该节点的子节点进 行处理，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)

比较如果都有子节点，则进行 updateChildren，判断如何对这些新 老节点的子节点进行操作（diff 核心）。

匹配时，找到相同的子节点，递归比较子节点

在 diff 中，只对同层的子节点进行比较，放弃跨级的节点比较，使 得时间复杂从 O(n 3)降低值 O(n)，也就是说，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。

## 39. Vue 中 key 的作用

vue 中 key 值的作用可以分为两种情况来考虑：

第一种情况是 v-if 中使用 key。由于 Vue 会尽可能高效地渲染元 素，通常会复用已有元素而不是从头开始渲染。因此当使用 v-if 来 实现元素切换的时候，如果切换前后含有相同类型的元素，那么这个 元素就会被复用。如果是相同的 input 元素，那么切换前后用户的 输入不会被清除掉，这样是不符合需求的。因此可以通过使用 key 来 唯一的标识一个元素，这个情况下，使用 key 的元素不会被复用。这个时候 key 的作用是用来标识一个独立的元素。

第二种情况是 v-for 中使用 key。用 v-for 更新已渲染过的元素列 表时，它默认使用“就地复用”的策略。如果数据项的顺序发生了改 变，Vue 不会移动 DOM 元素来匹配数据项的顺序，而是简单复用此 处的每个元素。因此通过为每个列表项提供一个 key 值，来以便 Vue 跟踪元素的身份，从而高效的实现复用。这个时候 key 的作用是为 了高效的更新渲染虚拟 DOM。

key 是为 Vue 中 vnode 的唯一标记，通过这个 key，diff 操作可 以更准确、更快速

更准确：因为带 key 就不是就地复用了，在 sameNode 函数 a.key === b.key 对比中可以避免就地复用的情况。所以会更加准确。

更快速：利用 key 的唯一性生成 map 对象来获取对应节点，比遍历 方式更快

## 40.如何从真实DOM到虚拟DOM

涉及到Vue中的模板编译原理，主要过程：

将模板转换成 ast 树， ast 用对象来描述真实的JS语法（将真实DOM转换成虚拟DOM）

优化树

将 ast 树生成代码

## 41.为什么Vue采用异步渲染呢？

Vue 是组件级更新，如果不采用异步更新，那么每次更新数据都会对当前组件进行重新渲染，所以为了性能， Vue 会在本轮数据更新后，在异步更新视图。核心思想 nextTick 。

dep.notify（） 通知 watcher进行更新， subs[i].update 依次调用 watcher 的 update ， queueWatcher 将watcher 去重放入队列， nextTick（ flushSchedulerQueue ）在下一tick中刷新watcher队列（异步）。

## 42.为什么vue组件中data必须是一个函数？

对象为引用类型，当复用组件时，由于数据对象都指向同一个data对象，当在一个组件中修改data时，其他重用的组件中的data会同时被修改；而使用返回对象的函数，由于每次返回的都是一个新对象（Object的实例），引用地址不同，则不会出现这个问题。

## 43.Vue-router 导航守卫有哪些

全局前置/钩子：beforeEach、beforeResolve、afterEach

路由独享的守卫：beforeEnter

组件内的守卫：beforeRouteEnter、beforeRouteUpdate、beforeRouteLeave

## 44.Vue 为什么要用 vm.$set() 解决对象新增属性不能响应的问题 ？你能说说如下代码的实现原理么？

### 1）Vue为什么要用vm.$set() 解决对象新增属性不能响应的问题

Vue使用了Object.defineProperty实现双向数据绑定

在初始化实例时对属性执行 getter/setter 转化

属性必须在data对象上存在才能让Vue将它转换为响应式的（这也就造成了Vue无法检测到对象属性的添加或删除）

所以Vue提供了Vue.set (object, propertyName, value) / vm.$set (object, propertyName, value)

### 2）接下来我们看看框架本身是如何实现的呢?

Vue 源码位置：vue/src/core/instance/index.js

export function set (target: Array<any> | Object, key: any, val: any): any {

// target 为数组

if (Array.isArray(target) && isValidArrayIndex(key)) {

// 修改数组的长度, 避免索引>数组长度导致splcie()执行有误

target.length = Math.max(target.length, key)

// 利用数组的splice变异方法触发响应式

target.splice(key, 1, val)

return val

}

// key 已经存在，直接修改属性值

if (key in target && !(key in Object.prototype)) {

target[key] = val

return val

}

const ob = (target: any).\_\_ob\_\_

// target 本身就不是响应式数据, 直接赋值

if (!ob) {

target[key] = val

return val

}

// 对属性进行响应式处理

defineReactive(ob.value, key, val)

ob.dep.notify()

return val

}

我们阅读以上源码可知，vm.$set 的实现原理是：

如果目标是数组，直接使用数组的 splice 方法触发相应式；

如果目标是对象，会先判读属性是否存在、对象是否是响应式，

最终如果要对属性进行响应式处理，则是通过调用 defineReactive 方法进行响应式处理

defineReactive 方法就是 Vue 在初始化对象时，给对象属性采用 Object.defineProperty 动态添加 getter 和 setter 的功能所调用的方法

## 说一说Vuex是什么，每个属性是干嘛的，如何使用 ？

得分点： state、mutations、getters、actions、module、store.commit、store.dispatch

标准回答 ：

Vuex是集中管理项目公共数据的。Vuex 有state、mutations 、getters、actions、module属性。

state 属性用来存储公共管理的数据。

mutations 属性定义改变state中数据的方法， 注意：不要在mutation中的方法中写异步方法ajax，那样数据就不可跟踪了。

getters 属性可以认为是定义 store 的计算属性。就像计算属性一样，getter的返回值会根据它的依赖被缓存起来，且只有当它的依赖值发生了改变才会被重新计算。

action属性类似于mutation，不同在于：Action 提交的是 mutation，而不是直接变更状态。Action 可以包含任意异步操作。

moudle属性是将store分割成模块。每个模块拥有自己的state、mutation、action、getter、甚至是嵌套子模块，从上至下进行同样方式的分割

使用方法：

state：直接以对象方式添加属性

mutations ：通过store.commit调用

action：通过 store.dispatch方法触发

getters：直接通过store.getters.调用

加分回答：可以使用辅助函数mapState、mapMutations、mapAction、mapGetters一次性获取每个属性下对应的多个方法。 VueX在大型项目中比较常用，非关系组件传递数据比较方便。