Vue 八股

1. 谈一谈对 *MVVM* 的理解?

> 参考答案:
>
> MVVM 是 Model-View-ViewModel 的缩写。MVVM 是一种设计思想。 Model 层代表数据模型，也可以在 Model 中定义数据修改和操作的业务逻辑; View 代表 UI 组件，它负责将数据模型转化成 UI 展现出来，View 是一个同步 View 和 Model 的对象 在 MVVM 架构下，View 和 Model 之间并没有直接的联系，而是通过 ViewModel 进行交互， Model 和 ViewModel 之间的交互是双向的， 因此 View 数据的变化会同步到 Model 中，而 Model 数据的变化也会立即反应到 View 上。 对 ViewModel 通过双向数据绑定把 View 层和 Model 层连接了起来，而 View 和 Model 之间的 同步工作完全是自动的，无需人为干涉，因此开发者只需关注业务逻辑，不需要手动操作 DOM，不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理。

2. 说一下 *Vue* 的优点

> 参考答案:
>
> Vue 是一个构建数据驱动的 Web 界面的渐进式框架。 Vue 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。核心是一个响应的数据绑定系统。关于 Vue 的优点，主要有响应式编程、组件化开发、虚拟 DOM
>
> - 响应式编程
>
> 这里的响应式不是 @media 媒体查询中的响应式布局，而是指 Vue 会自动对页面中某些数据的变化做出响应。这也就是 Vue 最大的优点，通过 MVVM 思想实现数据的双向绑定，让开发者不用再操作 DOM 对象，有更多的时间去思考业务逻辑。
>
> - 组件化开发

> Vue 通过组件，把一个单页应用中的各种模块拆分到一个一个单独的组件（component）中，我们只要先在父级应用中写好各种组件标签（占坑），并且在组件标签中写好要传入组件的参数（就像给函数传入参数一样，这个参数叫做组件的属性），然后再分别写好各种组件的实现（填坑），然后整个应用就算做完了。 组件化开发的优点：提高开发效率、方便重复使用、简化调试步骤、提升整个项目的可维护性、便于协同开发。
>
> - 虚拟 DOM
>
> 在传统开发中，用 JQuery 或者原生的 JavaScript DOM 操作函数对 DOM 进行频繁操作的时候，浏览器要不停的渲染新的 DOM 树，导致在性能上面的开销特别的高。而 Virtual DOM 则是虚拟 DOM 的英文，简单来说，他就是一种可以预先通过 JavaScript 进行各种计算，把最终的 DOM 操作计算出来并优化，由于这个 DOM 操作属于预处理操作，并没有真实的操作 DOM，所以叫做虚拟 DOM。最后在计算完毕才真正将 DOM 操作提交，将 DOM 操作变化反映到 DOM 树上。
>
> - 响应式编程
>
> 这里的响应式不是 @media 媒体查询中的响应式布局，而是指 Vue 会自动对页面中某些数据的变化做出响应。这也就是 Vue 最大的优点，通过 MVVM 思想实现数据的双向绑定，让开发者不用再操作 DOM 对象，有更多的时间去思考业务逻辑。
>
> 组件化开发
>
> Vue 通过组件，把一个单页应用中的各种模块拆分到一个一个单独的组件（component）中，我们只要先在父级应用中写好各种组件标签（占坑），并且在组件标签中写好要传入组件的参数（就像给函数传入参数一样，这个参数叫做组件的属性），然后再分别写好各种组件的实现（填坑），然后整个应用就算做完了。
>
> 组件化开发的优点：提高开发效率、方便重复使用、简化调试步骤、提升整个项目的可维护性、便于协同开发。

3. 解释一下对 *Vue* 生命周期的理解

可以从以下方面展开回答：

- 什么是 *vue* 生命周期

- *vue* 生命周期的作用是什么
- *vue* 生命周期有几个阶段
- 第一次页面加载会触发哪几个钩子
- *DOM* 渲染在哪个周期就已经完成
- 多组件（父子组件）中生命周期的调用顺序说一下

> 参考答案：
>
> - **什么是 vue 生命周期**
>
> 对于 vue 来讲，生命周期就是一个 vue 实例从创建到销毁的过程。
>
> - **vue 生命周期的作用是什么**
>
> 在生命周期的过程中会运行着一些叫做生命周期的函数，给予了开发者在不同的生命周期阶段添加业务代码的能力。
>
> 其实和回调是一个概念，当系统执行到某处时，检查是否有 hook(钩子)，有的话就会执行回调。
>
> 通俗的说，hook 就是在程序运行中，在某个特定的位置，框架的开发者设计好了一个钩子来告诉我们当前程序已经运行到特定的位置了，会触发一个回调函数，并提供给我们，让我们可以在生命周期的特定阶段进行相关业务代码的编写。
>
> - **vue 生命周期有几个阶段**
>
> 它可以总共分为 8 个阶段：创建前/后, 载入前/后,更新前/后,销毁前/销毁后。
>
> 1. 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。
> 2. 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 。

3. 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。
4. 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。
5. 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。
6. 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。
7. 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。
8. 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。

- **第一次页面加载会触发哪几个钩子**

会触发 4 个钩子，分别是：beforeCreate、created、beforeMount、mounted

- **DOM 渲染在哪个周期就已经完成**

DOM 渲染是在 mounted 阶段完成，此阶段真实的 DOM 挂载完毕，数据完成双向绑定，可以访问到 DOM 节点。

- **多组件（父子组件）中生命周期的调用顺序说一下**

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。

1. 加载渲染过程：父beforeCreate->父created->父beforeMount->子beforeCreate->子created->子beforeMount- >子mounted->父mounted
2. 子组件更新过程：父beforeUpdate->子beforeUpdate->子updated->父updated
3. 父组件更新过程：父 beforeUpdate -> 父 updated
4. 销毁过程：父beforeDestroy->子beforeDestroy->子destroyed->父destroyed

4. *Vue* 实现双向数据绑定原理是什么?

> 参考答案:
>
> Vue2.x 采用数据劫持结合发布订阅模式（PubSub 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。
>
> 当把一个普通 Javascript 对象传给 Vue 实例来作为它的 data 选项时，Vue 将遍历它的属性，用 Object.defineProperty 将它们转为 getter/setter。用户看不到 getter/setter，但是在内部它们让 Vue 追踪依赖，在属性被访问和修改时通知变化。
>
> Vue 的数据双向绑定整合了 Observer，Compile 和 Watcher 三者，通过 Observer 来监听自己的 model 的数据变化，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（例如 input 操作）->数据 model 变更的双向绑定效果。
>
> Vue3.x 放弃了 Object.defineProperty ，使用 ES6 原生的 Proxy，来解决以前使用 Object.defineProperty 所存在的一些问题。

5. 说一下对 *Vue2.x* 响应式原理的理解

> 参考答案:
>
> 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)。
>
> （可以参阅前面第 4 题答案）

6. 说一下在 *Vue2.x* 中如何检测数组的变化?

> 参考答案:

Vue2.x 中实现检测数组变化的方法，是将数组的常用方法进行了重写。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。

流程:

1. 初始化传入 data 数据执行 initData
2. 将数据进行观测 new Observer
3. 将数组原型方法指向重写的原型
4. 深度观察数组中的引用类型

有两种情况无法检测到数组的变化。

1. 当利用索引直接设置一个数组项时，例如 vm.items[indexOfItem] = newValue
2. 当修改数组的长度时，例如 vm.items.length = newLength

不过这两种场景都有对应的解决方案。

利用索引设置数组项的替代方案

```
//使用该方法进行更新视图

// vm.$set，Vue.set的一个别名

vm.$set(vm.items, indexOfItem, newValue)
```

修改数组的长度的替代方案

```
//使用该方法进行更新视图

// Array.prototype.splice

vm.items.splice(indexOfItem, 1, newValue)
```

7. *Vue3.x* 响应式数据

可以从以下方面展开回答：

- *Vue3.x* 响应式数据原理是什么?
- *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?
- 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?

> 参考答案：
>
> **Vue3.x 响应式数据原理是什么?**
>
> 在 Vue 2 中，响应式原理就是使用的 Object.defineProperty 来实现的。但是在 Vue 3.0 中采用了 Proxy，抛弃了 Object.defineProperty 方法。
>
> 究其原因，主要是以下几点：
>
> 1. Object.defineProperty 无法监控到数组下标的变化，导致通过数组下标添加元素，不能实时响应
> 2. Object.defineProperty 只能劫持对象的属性，从而需要对每个对象，每个属性进行遍历，如果，属性值是对象，还需要深度遍历。Proxy 可以劫持整个对象，并返回一个新的对象。
> 3. Proxy 不仅可以代理对象，还可以代理数组。还可以代理动态增加的属性。
> 4. Proxy 有多达 13 种拦截方法
> 5. Proxy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。
>
> **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。

8. *v-model* 双向绑定的原理是什么?

> 参考答案:
>
> v-model 本质就是 :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。
>
> 例如:
>
> 1. text 和 textarea 元素使用 value 属性和 input 事件
> 2. checkbox 和 radio 使用 checked 属性和 change 事件
> 3. select 字段将 value 作为 prop 并将 change 作为事件
>
> 以输入框为例，当用户在输入框输入内容时，会触发 input 事件，从而更新 value。而 value 的改变同样会更新视图，这就是 vue 中的双向绑定。双向绑定的原理，其实现思路如下:
>
> 首先要对数据进行劫持监听，所以我们需要设置一个监听器 Observer，用来监听所有属性。如果属性发上变化了，就需要告诉订阅者 Watcher 看是否需要更新。
>
> 因为订阅者是有很多个，所以我们需要有一个消息订阅器 Dep 来专门收集这些订阅者，然后在监听器 Observer 和订阅者 Watcher 之间进行统一管理的。
>
> 接着，我们还需要有一个指令解析器 Compile，对每个节点元素进行扫描和解析，将相关指令对应初始化成一个订阅者 Watcher，并替换模板数据或者绑定相应的函数，此时当订阅者 Watcher 接收到相应属性的变化，就会执行对应的更新函数，从而更新视图。
>
> 因此接下去我们执行以下 3 个步骤，实现数据的双向绑定:
>
> 1. 实现一个监听器 Observer，用来劫持并监听所有属性，如果有变动的，就通知订阅者。
> 2. 实现一个订阅者 Watcher，可以收到属性的变化通知并执行相应的函数，从而更新视图。

3. 实现一个解析器 Compile，可以扫描和解析每个节点的相关指令，并根据初始化模板数据以及初始化相应的订阅器。

9. *vue2.x* 和 *vuex3.x* 渲染器的 *diff* 算法分别说一下?

参考答案：

简单来说，diff 算法有以下过程

- 同级比较，再比较子节点
- 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)
- 比较都有子节点的情况(核心 diff)
- 递归比较子节点

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) -> 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。该算法中还运用了动态规划的思想求解最长递归子序列。

10. *vue* 组件的参数传递

可以从以下方面展开回答：

- 解释一下父组件与子组件传值实现过程
- 非父子组件的数据传递，兄弟组件传值是如何实现的

参考答案：

解释一下父组件与子组件传值实现过程

- 父组件传给子组件：子组件通过 props 方法接受数据
- 子组件传给父组件：使用自定义事件，自组件通过 $emit 方法触发父组件的方法来传递参数

非父子组件的数据传递，兄弟组件传值是如何实现的

eventBus，就是创建一个事件中心，相当于中转站，可以用它来传递事件和接收事件。项目比较小时，用这个比较合适。

此外，总结 vue 中的组件通信方式，常见使用场景可以分为三类：

- 父子通信：

父向子传递数据是通过 props ， 子向父是通过 on

bus

- vuex

通过父链 / 子链也可以通信（ children ）

ref 也可以访问组件实例

- v-model
- sync 修饰符
- 兄弟通信：

bus

- vuex
- 跨级通信：

> bus
>
> vuex
>
> provide / inject API
>
> listeners

11. *Vue* 的路由实现

可以从以下方面展开回答：

- 解释 *hash* 模式和 *history* 模式的实现原理
- 说一下 [?] *route* 的区别
- *vueRouter* 有哪几种导航守卫?
- 解释一下 *vueRouter* 的完整的导航解析流程是什么

> 参考答案：
>
> **解释 hash 模式和 history 模式的实现原理**
>
> 后面 hash 值的变化，不会导致浏览器向服务器发出请求，浏览器不发出请求，就不会刷新页面；通过监听 hashchange 事件可以知道 hash 发生了哪些变化，然后根据 hash 变化来实现更新页面部分内容的操作。
>
> history 模式的实现，主要是 HTML5 标准发布的两个 API，pushState 和 replaceState，这两个 API 可以在改变 URL，但是不会发送请求。这样就可以监听 url 变化来实现更新页面部分内容的操作。
>
> 两种模式的区别：
>
> - 首先是在 URL 的展示上，hash 模式有“#”，history 模式没有
> - 刷新页面时，hash 模式可以正常加载到 hash 值对应的页面，而 history 没有处理的话，会返回 404，一般需要后端将所有页面都配置重定向到首页路由
> - 在兼容性上，hash 可以支持低版本浏览器和 IE

说一下 [?] route 的区别

$route 对象表示当前的路由信息，包含了当前 URL 解析得到的信息。包含当前的路径，参数，query 对象等。

- $route.params： 一个 key/value 对象，包含了 动态片段 和 全匹配片段，如果没有路由参数，就是一个空对象。
- route.query.user == 1，如果没有查询参数，则是个空对象。
- $route.hash：当前路由的 hash 值 (不带 #) ，如果没有 hash 值，则为空字符串。
- $route.fullPath：完成解析后的 URL，包含查询参数和 hash 的完整路径。
- $route.matched：数组，包含当前匹配的路径中所包含的所有片段所对应的配置参数对象。
- $route.name：当前路径名字
- $route.meta：路由元信息

$route 对象出现在多个地方:

- 组件内的 this.$route 和 route watcher 回调（监测变化处理）
- router.match(location) 的返回值
- scrollBehavior 方法的参数
- 导航钩子的参数，例如 router.beforeEach 导航守卫的钩子函数中，to 和 from 都是这个路由信息对象。

$router 对象是全局路由的实例，是 router 构造方法的实例。

$router 对象常用的方法有：

- push：向 history 栈添加一个新的记录
- go：页面路由跳转前进或者后退
- replace：替换当前的页面，不会向 history 栈添加一个新的记录

vueRouter 有哪几种导航守卫?

- 全局前置/钩子：beforeEach、beforeR-esolve、afterEach
- 路由独享的守卫：beforeEnter
- 组件内的守卫：beforeRouteEnter、beforeRouteUpdate、beforeRouteLeave

解释一下 vueRouter 的完整的导航解析流程是什么

一次完整的导航解析流程如下：

- 1.导航被触发。
- 2.在失活的组件里调用离开守卫。
- 3.调用全局的 beforeEach 守卫。
- 4.在重用的组件里调用 beforeRouteUpdate 守卫（2.2+）。
- 5.在路由配置里调用 beforeEnter。
- 6.解析异步路由组件。
- 7.在被激活的组件里调用 beforeRouteEnter。
- 8.调用全局的 beforeResolve 守卫（2.5+）。
- 9.导航被确认。
- 10.调用全局的 afterEach 钩子。
- 11.触发 DOM 更新。
- 12.用创建好的实例调用 beforeRouteEnter 守卫中传给 next 的回调函数。

12. *vuex* 是什么？怎么使用它？什么场景下我们会使用到 *vuex*

参考答案：

vuex 是什么

vuex 是一个专为 Vue 应用程序开发的状态管理器，采用集中式存储管理应用的所有组件的状态。每一个 vuex 应用的核心就是 store（仓库）。“store” 基本上就是一个容器，它包含着应用中大部分的状态 (state)。

为什么需要 vuex

由于组件只维护自身的状态(data)，组件创建时或者路由切换时，组件会被初始化，从而导致 data 也随之销毁。

使用方法

在 main.js 引入 store，注入。只用来读取的状态集中放在 store 中， 改变状态的方式是提交 mutations，这是个同步的事物，异步逻辑应该封装在 action 中。

什么场景下会使用到 vuex

如果是 vue 的小型应用，那么没有必要使用 vuex，这个时候使用 vuex 反而会带来负担。组件之间的状态传递使用 props、自定义事件来传递即可。

但是如果涉及到 vue 的大型应用，那么就需要类似于 vuex 这样的集中管理状态的状态机来管理所有组件的状态。例如登录状态、加入购物车、音乐播放等，总之只要是开发 vue 的大型应用，都推荐使用 vuex 来管理所有组件状态。

13. 说一下 *v-if* 与 *v-show* 的区别

参考答案：

- 共同点：都是动态显示 DOM 元素
- 区别点:

1、手段

v-if 是动态的向 DOM 树内添加或者删除 DOM 元素

v-show 是通过设置 DOM 元素的 display 样式属性控制显隐

2、编译过程

v-if 切换有一个局部编译/卸载的过程，切换过程中合适地销毁和重建内部的事件监听和子组件

v-show 只是简单的基于 css 切换

3、编译条件

v-if 是惰性的，如果初始条件为假，则什么也不做。只有在条件第一次变为真时才开始局部编译

v-show 是在任何条件下(首次条件是否为真)都被编译，然后被缓存，而且 DOM 元素保留

4、性能消耗

v-if 有更高的切换消耗

v-show 有更高的初始渲染消耗

5、使用场景

v-if 适合运营条件不大可能改变

v-show 适合频繁切换

14. 如何让 *CSS* 值在当前的组件中起作用

参考答案：

在 vue 文件中的 style 标签上，有一个特殊的属性：scoped。当一个 style 标签拥有 scoped 属性时，它的 CSS 样式就只能作用于当前的组件，也就是说，该样式只能适用于当前组件元素。通过该属性，可以使得组件之间的样式不互相污染。如果一个项目中的所有 style 标签全部加上了 scoped，相当于实现了样式的模块化。

scoped 的实现原理

vue 中的 scoped 属性的效果主要通过 PostCSS 转译实现的。PostCSS 给一个组件中的所有 DOM 添加了一个独一无二的动态属性，然后，给 CSS 选择器额外添加一个对应的属性选择器来选择该组件中 DOM，这种做法使得样式只作用于含有该属性的 DOM，即组件内部 DOM。

例如：

转译前

```
<template>

<div class="example">hi</div>

</template>



<style scoped>

.example {

color: red;

}

</style>
```

转译后：

```
<template>

<div class="example" data-v-5558831a>hi</div>

</template>



<style>

.example[data-v-5558831a] {

color: red;
```

```
}

</style>
```

15. *keep-alive* 相关

- keep-alive的实现原理是什么
- 与keep-alive相关的生命周期函数是什么，什么场景下会进行使用
- keep-alive的常用属性有哪些

> 参考答案：
>
> keep-alive 组件是 vue 的内置组件，用于缓存内部组件实例。这样做的目的在于，keep-alive 内部的组件切回时，不用重新创建组件实例，而直接使用缓存中的实例，一方面能够避免创建组件带来的开销，另一方面可以保留组件的状态。
>
> keep-alive 具有 include 和 exclude 属性，通过它们可以控制哪些组件进入缓存。另外它还提供了 max 属性，通过它可以设置最大缓存数，当缓存的实例超过该数时，vue 会移除最久没有使用的组件缓存。
>
> 受keep-alive的影响，其内部所有嵌套的组件都具有两个生命周期钩子函数，分别是 activated 和 deactivated，它们分别在组件激活和失活时触发。第一次 activated 触发是在 mounted 之后
>
> 在具体的实现上，keep-alive 在内部维护了一个 key 数组和一个缓存对象

```
// keep-alive 内部的声明周期函数

created () {

 this.cache = Object.create(null)

 this.keys = []

}
```

key 数组记录目前缓存的组件 key 值，如果组件没有指定 key 值，则会为其自动生成一个唯一的 key 值

cache 对象以 key 值为键，vnode 为值，用于缓存组件对应的虚拟 DOM

在 keep-alive 的渲染函数中，其基本逻辑是判断当前渲染的 vnode 是否有对应的缓存，如果有，从缓存中读取到对应的组件实例；如果没有则将其缓存。

当缓存数量超过 max 数值时，keep-alive 会移除掉 key 数组的第一个元素。

16. *Vue* 中如何进行组件的使用？*Vue* 如何实现全局组件的注册？

参考答案：

要使用组件，首先需要使用 import 来引入组件，然后在 components 属性中注册组件，之后就可以在模板中使用组件了。

可以使用 Vue.component 方法来实现全局组件的注册。

17. *vue-cli* 工程相关

- 构建 *vue-cli* 工程都用到了哪些技术？他们的作用分别是什么？
- *vue-cli* 工程常用的 *npm* 命令有哪些？

参考答案：

构建 vue-cli 工程都用到了哪些技术？他们的作用分别是什么？

- vue.js：vue-cli 工程的核心，主要特点是双向数据绑定和组件系统。
- vue-router：vue 官方推荐使用的路由框架。
- vuex：专为 Vue.js 应用项目开发的状态管理器，主要用于维护 vue 组件间共用的一些 变量 和 方法。
- axios（或者 fetch、ajax）：用于发起 GET 、或 POST 等 http请求，基于 Promise 设计。
- vux等：一个专为vue设计的移动端UI组件库。
- webpack：模块加载和vue-cli工程打包器。
- eslint：代码规范工具

vue-cli 工程常用的 npm 命令有哪些?

- 下载 node_modules 资源包的命令：npm install
- 启动 vue-cli 开发环境的 npm命令：npm run dev
- vue-cli 生成 生产环境部署资源 的 npm命令：npm run build
- 用于查看 vue-cli 生产环境部署资源文件大小的 npm命令：npm run build --report

18. *nextTick* 的作用是什么？他的实现原理是什么?

参考答案：

作用：vue 更新 DOM 是异步更新的，数据变化，DOM 的更新不会马上完成，nextTick 的回调是在下次 DOM 更新循环结束之后执行的延迟回调。

实现原理：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

- Promise：可以将函数延迟到当前函数调用栈最末端
- MutationObserver ： 是 H5 新加的一个功能，其功能是监听 DOM 节点的变动，在所有 DOM 变动完成后，执行回调函数

- setImmediate：用于中断长时间运行的操作，并在浏览器完成其他操作（如事件和显示更新）后立即运行回调函数
- 如果以上都不行则采用 setTimeout 把函数延迟到 DOM 更新之后再使用，原因是宏任务消耗大于微任务，优先使用微任务，最后使用消耗最大的宏任务。

19. 说一下 *Vue SSR* 的实现原理

参考答案：

- app.js 作为客户端与服务端的公用入口，导出 Vue 根实例，供客户端 entry 与服务端 entry 使用。客户端 entry 主要作用挂载到 DOM 上，服务端 entry 除了创建和返回实例，还需要进行路由匹配与数据预获取。
- webpack 为客服端打包一个 ClientBundle，为服务端打包一个 ServerBundle。
- 服务器接收请求时，会根据 url，加载相应组件，获取和解析异步数据，创建一个读取 Server Bundle 的 BundleRenderer，然后生成 html 发送给客户端。
- 客户端混合，客户端收到从服务端传来的 DOM 与自己的生成的 DOM 进行对比，把不相同的 DOM 激活，使其可以能够响应后续变化，这个过程称为客户端激活（也就是转换为单页应用）。为确保混合成功，客户 端与服务器端需要共享同一套数据。在服务端，可以在渲染之前获取数据，填充到 store 里，这样，在客户端挂载到 DOM 之前，可以直接从 store 里取数据。首屏的动态数据通过 window.*INITIAL_STATE* 发送到客户端
- VueSSR 的原理，主要就是通过 vue-server-renderer 把 Vue 的组件输出成一个完整 HTML，输出到客户端，到达客户端后重新展开为一个单页应用。

20. *Vue* 组件的 *data* 为什么必须是函数

参考答案：

组件中的 data 写成一个函数，数据以函数返回值形式定义。这样每复用一次组件，就会返回一份新的 data，类似于给每个组件实例创建一个私有的数据空间，让各个组件实例维护各自的数据。而单纯的写成对象形式，就使得所有组件实例共用了一份 data，就会造成一个变了全都会变的结果。

21. 说一下 *Vue* 的 *computed* 的实现原理

> 参考答案：
>
> 当组件实例触发生命周期函数 beforeCreate 后，它会做一系列事情，其中就包括对 computed 的处理。
>
> 它会遍历 computed 配置中的所有属性，为每一个属性创建一个 Watcher 对象，并传入一个函数，该函数的本质其实就是 computed 配置中的 getter，这样一来，getter 运行过程中就会收集依赖
>
> 但是和渲染函数不同，为计算属性创建的 Watcher 不会立即执行，因为要考虑到该计算属性是否会被渲染函数使用，如果没有使用，就不会得到执行。因此，在创建 Watcher 的时候，它使用了 lazy 配置，lazy 配置可以让 Watcher 不会立即执行。
>
> 收到 lazy 的影响，Watcher 内部会保存两个关键属性来实现缓存，一个是 value，一个是 dirty
>
> value 属性用于保存 Watcher 运行的结果，受 lazy 的影响，该值在最开始是 undefined
>
> dirty 属性用于指示当前的 value 是否已经过时了，即是否为脏值，受 lazy 的影响，该值在最开始是 true
>
> Watcher 创建好后，vue 会使用代理模式，将计算属性挂载到组件实例中
>
> 当读取计算属性时，vue 检查其对应的 Watcher 是否是脏值，如果是，则运行函数，计算依赖，并得到对应的值，保存在 Watcher 的 value 中，然后设置 dirty 为 false，然后返回。
>
> 如果 dirty 为 false，则直接返回 watcher 的 value
>
> 巧妙的是，在依赖收集时，被依赖的数据不仅会收集到计算属性的 Watcher，还会收集到组件的 Watcher
>
> 当计算属性的依赖变化时，会先触发计算属性的 Watcher 执行，此时，它只需设置 dirty 为 true 即可，不做任何处理。

> 由于依赖同时会收集到组件的 Watcher，因此组件会重新渲染，而重新渲染时又读取到了计算属性，由于计算属性目前已为 dirty，因此会重新运行 getter 进行运算
>
> 而对于计算属性的 setter，则极其简单，当设置计算属性时，直接运行 setter 即可。

22. 说一下 *Vue complier* 的实现原理是什么样的?

> 参考答案：
>
> 在使用 vue 的时候，我们有两种方式来创建我们的 HTML 页面，第一种情况，也是大多情况下，我们会使用模板 template 的方式，因为这更易读易懂也是官方推荐的方法；第二种情况是使用 render 函数来生成 HTML，它比 template 更接近最终结果。
>
> complier 的主要作用是解析模板，生成渲染模板的 render， 而 render 的作用主要是为了生成 VNode
>
> complier 主要分为 3 大块：
>
> - parse：接受 template 原始模板，按着模板的节点和数据生成对应的 ast
> - optimize：遍历 ast 的每一个节点，标记静态节点，这样就知道哪部分不会变化，于是在页面需要更新时，通过 diff 减少去对比这部分DOM，提升性能
> - generate 把前两步生成完善的 ast，组成 render 字符串，然后将 render 字符串通过 new Function 的方式转换成渲染函数

23. *vue* 如何快速定位那个组件出现性能问题的

> 参考答案：
>
> 用 timeline 工具。 通过 timeline 来查看每个函数的调用时常，定位出哪个函数的问题，从而能判断哪个组件出了问题。

24. *Proxy* 相比 *defineProperty* 的优势在哪里

> 参考答案：
>
> Vue3.x 改用 Proxy 替代 Object.defineProperty
>
> 原因在于 Object.defineProperty 本身存在的一些问题：
>
> - Object.defineProperty 只能劫持对象属性的 getter 和 setter 方法。
> - Object.definedProperty 不支持数组(可以监听数组,不过数组方法无法监听自己重写)，更准确的说是不支持数组的各种 API(所以 Vue 重写了数组方法。
>
> 而相比 Object.defineProperty，Proxy 的优点在于：
>
> - Proxy 是直接代理劫持整个对象。
> - 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。
>
> 目前，Object.definedProperty 唯一比 Proxy 好的一点就是兼容性，不过 Proxy 新标准也受到浏览器厂商重点持续的性能优化当中。

25. *Vue* 与 *Angular* 以及 *React* 的区别是什么?

> 参考答案：
>
> 这种题目是开放性题目，一般是面试过程中面试官口头来提问，不太可能出现在笔试试卷里面。
>
> 关于 Vue 和其他框架的不同，官方专门写了一篇文档，从性能、体积、灵活性等多个方面来进行了说明。
>
> 详细可以参阅：cn.vuejs.org/v2/guide/co…
>
> 建议面试前通读一遍该篇文档，然后进行适当的总结。

26. 说一下 *watch* 与 *computed* 的区别是什么？以及他们的使用场景分别是什么?

> 参考答案：
>
> 区别：
>
> - 都是观察数据变化的（相同）
> - 计算属性将会混入到 vue 的实例中，所以需要监听自定义变量；watch 监听 data 、props 里面数据的变化；
> - computed 有缓存，它依赖的值变了才会重新计算，watch 没有；
> - watch 支持异步，computed 不支持；
> - watch 是一对多（监听某一个值变化，执行对应操作）；computed 是多对一（监听属性依赖于其他属性）
> - watch 监听函数接收两个参数，第一个是最新值，第二个是输入之前的值；
> - computed 属性是函数时，都有 get 和 set 方法，默认走 get 方法，get 必须有返回值（return）
>
> watch 的 参数：
>
> - deep：深度监听
> - immediate： 组件加载立即触发回调函数执行
>
> computed 缓存原理：
>
> conputed本质是一个惰性的观察者；当计算数据存在于 data 或者 props里时会被警告；
>
> vue 初次运行会对 computed 属性做初始化处理（initComputed），初始化的时候会对每一个 computed 属性用 watcher 包装起来 ，这里面会生成一个 dirty 属性值为 true；然后执行 defineComputed 函数来计算，计算之后会将 dirty 值变为 false，这里会根据 dirty 值来判断是否需要重新计算；如果属性依赖的数据发生变化，computed 的 watcher 会把 dirty 变为 true，这样就会重新计算 computed 属性的值。

27. *scoped* 是如何实现样式穿透的?

参考答案:

首先说一下什么场景下需要 scoped 样式穿透。

在很多项目中，会出现这么一种情况，即：引用了第三方组件，需要在组件中局部修改第三方组件的样式，而又不想去除 scoped 属性造成组件之间的样式污染。此时只能通过特殊的方式，穿透 scoped。

有三种常用的方法来实现样式穿透。

方法一

使用 ::v-deep 操作符(>>> 的别名)

如果希望 scoped 样式中的一个选择器能够作用得“更深”，例如影响子组件，可以使用 >>> 操作符:

```
<style scoped>

 .a >>> .b { /* ... */ }

</style>
```

上述代码将会编译成:

```
.a[data-v-f3f3eg9] .b { /* ... */ }
```

后面的类名没有 data 属性，所以能选到子组件里面的类名。

有些像 Sass 之类的预处理器无法正确解析 >>>，所以需要使用 ::v-deep 操作符来代替。

方法二

定义一个含有 scoped 属性的 style 标签之外，再定义一个不含有 scoped 属性的 style 标签，即在一个 vue 组件中定义一个全局的 style 标签，一个含有作用域的 style 标签：

```
<style>

/* global styles */

</style>



<style scoped>

/* local styles */

</style>
```

此时，我们只需要将修改第三方样式的 css 写在第一个 style 中即可。

方法三

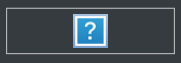
上面的方法一需要单独书写一个不含有 scoped 属性的 style 标签，可能会造成全局样式的污染。

更推荐的方式是在组件的外层 DOM 上添加唯一的 class 来区分不同组件，在书写样式时就可以正常针对针对这部

vue3比vue2有什么优势?

性能更好，打包体积更小，更好的ts支持，更好的代码组织，更好的逻辑抽离，更多的新功能

描述Vu3生命周期



Options API的生命周期:

1. `beforeCreate`：在实例初始化之后、数据观测(initState)和 event/watcher 事件配置之前被调用。 对于此时做的事情，如注册组件使用到的store或者service等单例的全局物件。相比Vue2没有变化。
2. `created`：一个新的 Vue 实例被创建后（包括组件实例），立即调用此函数。 在这里做一下初始的数据处理、异步请求等操作，当组件完成创建时就能展示这些数据。 相比Vue2没有变化。
3. `beforeMount`：在挂载之前调用，相关的render函数首次被调用,在这里可以访问根节点，在执行mounted钩子前，dom渲染成功，相对Vue2改动不明显。
4. `onMounted`：在挂载后调用，也就是所有相关的DOM都已入图，有了相关的DOM环境，可以在这里执行节点的DOM操作。在这之前执行beforeUpdate。
5. `beforeUpdate`：在数据更新时同时在虚拟DOM重新渲染和打补丁之前调用。我们可以在这里访问先前的状态和dom，如果我们想要在更新之前保存状态的快照，这个钩子非常有用。相比Vue2改动不明显。
6. `onUpdated`:在数据更新完毕后，虚拟DOM重新渲染和打补丁也完成了，DOM已经更新完毕。这个钩子函数调用时，组件DOM已经被更新，可以执行操作，触发组件动画等操作
7. `beforeUnmount`:在卸载组件之前调用。在这里执行清除操作，如清除定时器、解绑全局事件等。
8. `onUnmounted`:在卸载组件之后调用，调用时，组件的DOM结构已经被拆卸，可以释放组件用过的资源等操作。

- `onActivated` – 被 `keep-alive` 缓存的组件激活时调用。
- `onDeactivated` – 被 `keep-alive` 缓存的组件停用时调用。

- `onErrorCaptured` – 当捕获一个来自子孙组件的错误时被调用。此钩子会收到三个参数：错误对象、发生错误的组件实例以及一个包含错误来源信息的字符串。此钩子可以返回 `false` 以阻止该错误继续向上传播。

Composition API的生命周期：

除了 `beforecate` 和 `created` (它们被 `setup` 方法本身所取代)，我们可以在 `setup` 方法中访问的上面后面9个生命钩子选项:

如何看待Composition API 和 Options API?

Composition API和Options API是Vue.js中的两种组件编写方式。

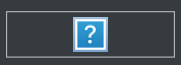
Options API是Vue.js早期版本中使用的编写方式，通过定义一个options对象进行组件的配置，包括props、data、methods、computed、watch等选项。这种方式的优点在于结构清晰、易于理解，在小型项目中比较实用。

Composition API是Vue.js 3.x版本中新引入的一种组件编写方式，它以函数的形式组织我们的代码，允许我们将相关部分组合起来，提高了代码的可维护性和重用性。Composition API还提供了模块化、类型推断等功能，可以更好地实现面向对象编程的思想。

Composition API 更好的代码组织，更好的逻辑服用；可维护性，更好的类型推导，可拓展性更好；

两种API各有优缺点，使用哪种API取决于具体的项目需求。对于小型项目，Options API更为简单方便；对于大型项目，Composition API可以更好地组织代码。

总之，Vue.js的Composition API和Options API是为了满足不同开发者的需求而存在的，我们应该根据具体的场景选择使用哪种API，以达到更好的开发效果和代码质量。



Vue3.0有什么更新

1. 性能优化：Vue.js 3.0使用了**Proxy**替代Object.defineProperty实现响应式，并且使用了静态提升技术来提高渲染性能。新增了编译时优化，在编译时进行模板静态分析，并生成更高效的渲染函数。

2. Composition API：Composition API是一个全新的组件逻辑复用方式，可以更好地组合和复用组件的逻辑。
3. TypeScript支持：Vue.js 3.0完全支持TypeScript，在编写Vue应用程序时可以更方便地利用TS的类型检查和自动补全功能。
4. 新的自定义渲染API：Vue.js 3.0的自定义渲染API允许开发者在细粒度上控制组件渲染行为，包括自定义渲染器、组件事件和生命周期等。
5. 改进的Vue CLI：Vue.js 3.0使用了改进的Vue CLI，可以更加灵活地配置项目，同时支持Vue.js2.x项目升级到Vue.js 3.0。
6. 移除一些API：Vue.js 3.0移除了一些不常用的API，如过渡相关API，部分修饰符等。

Proxy和Object.defineProperty的区别?

Proxy和Object.defineProperty都可以用来实现JavaScript对象的响应式，但是它们有一些区别：

1. 实现方式：Proxy是ES6新增的一种特性，使用了一种代理机制来实现响应式。而Object.defineProperty是在ES5中引入的，使用了getter和setter方法来实现。
2. 作用对象：Proxy可以代理**整个对象**，包括对象的所有属性、数组的所有元素以及类似数组对象的所有元素。而Object.defineProperty**只能代理对象上定义的属性**。
3. 监听属性：Proxy可以监听到新增属性和删除属性的操作，而Object.defineProperty**只能监听到已经**定义的属性的变化。
4. 性能：由于Proxy是ES6新增特性，其内部实现采用了更加高效的算法，相对于Object.defineProperty来说在性能方面有一定的优势。

综上所述，虽然Object.defineProperty在Vue.js 2.x中用来实现响应式，但是在Vue.js 3.0中已经采用了Proxy来替代，这是因为Proxy相对于Object.defineProperty拥有更优异的性能和更强大的能力。

Vue3升级了哪些重要功能?

- 新的API：Vue3使用createApp方法来创建应用程序实例，并有新的组件注册和调用方法。
- emits属性：： Vue 3的组件可以使用emits属性来声明事件。
- 生命周期

- 多个Fragment
- 移除.sync
- 异步组件的写法

```
const Foo = defineAsyncComponent(() => import('./Foo.vue') )
```

vue2和vue3 核心 diff 算法区别?

Vue 2.x使用的是双向指针遍历的算法，也就是通过逐层比对新旧虚拟DOM树节点的方式来计算出更新需要做的最小操作集合。但这种算法的缺点是，由于遍历是从左到右、从上到下进行的，当发生节点删除或移动时，会导致其它节点位置的计算出现错误，因此会造成大量无效的重新渲染。

Vue 3.x使用了经过优化的单向遍历算法，也就是只扫描新虚拟DOM树上的节点，判断是否需要更新，跳过不需要更新的节点，进一步减少了不必要的操作。此外，在虚拟DOM创建后，Vue 3会缓存虚拟DOM节点的描述信息，以便于复用，这也会带来性能上的优势。同时，Vue 3还引入了静态提升技术，在编译时将一些静态的节点及其子节点预先处理成HTML字符串，大大提升了渲染性能。

因此，总体来说，Vue 3相对于Vue 2拥有更高效、更智能的diff算法，能够更好地避免不必要的操作，并提高了渲染性能。

Vue3为什么比Vue2快?

1. 响应式系统优化：Vue3引入了新的响应式系统，这个系统的设计让Vue3的渲染函数可以在编译时生成更少的代码，这也就意味着在运行时需要更少的代码来处理虚拟DOM。这个新系统的一个重要改进就是提供了一种基于Proxy实现的响应式机制，这种机制为开发人员提供更加高效的API，也减少了一些运行时代码。
2. 编译优化：Vue3的编译器对代码进行了优化，包括减少了部分注释、空白符和其他非必要字符的编译，同时也对编译后的代码进行了懒加载优化。
3. 更快的虚拟DOM：Vue3对虚拟DOM进行了优化，使用了跟React类似的Fiber算法，这样可以更加高效地更新DOM节点，提高性能。
4. Composition API：Vue3引入了Composition API，这种API通过提供逻辑组合和重用的方法来提升代码的可读性和重用性。这种API不仅可以让Vue3应用更好地组织和维护业务逻辑，还可以让开发人员更加轻松地实现优化。

Vue3如何实现响应式?

使用Proxy和Reflect API实现vue3响应式。

Reflect API则可以更加方便地实现对对象的监听和更新，可以用来访问、检查和修改对象的属性和方法，比如 `Reflect.get` 、 `Reflect.set` 、 `Reflect.has` 等。

Vue3会将响应式对象转换为一个Proxy对象，并利用Proxy对象的get和set拦截器来实现对属性的监听和更新。当访问响应式对象的属性时，get拦截器会被触发，此时会收集当前的依赖关系，并返回属性的值；当修改响应式对象的属性时，set拦截器会被触发，此时会触发更新操作，并通知相关的依赖进行更新。

优点：可监听属性的变化、新增与删除，监听数组的变化

vue3.0编译做了哪一些优化?

Vue 3.0作为Vue.js的一次重大升级，其编译器也进行了一些优化，主要包括以下几方面：

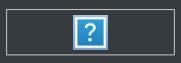
1. 静态树提升： Vue 3.0 通过重写编译器，实现对静态节点（即不改变的节点）进行编译优化，使用HoistStatic功能将静态节点移动到 render 函数外部进行缓存，从而服务端渲染和提高前端渲染的性能。
2. Patch Flag：在Vue 3.0中，编译的生成vnode会根据节点patch的标记，只对需要重新渲染的数据进行响应式更新，不需要更新的数据不会重新渲染，从而大大提高了渲染性能。
3. 静态属性提升：Vue3中对 `不参与更新` 的元素，会做静态提升， `只会被创建一次` ，在渲染时直接复用。免去了重复的创建操作，优化内存。 没做静态提升之前，未参与更新的元素也在 `render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` 。
4. `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` 。
5. 优化Render function：Vue 3.0的compile优化还包括：Render函数的换行和缩进、Render函数的条件折叠、Render函数的常量折叠等等。

总之，Vue 3.0通过多方面的编译优化，进一步提高了框架的性能和效率，使得Vue.js更加高效和易用。

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)=>{
  console.log("person.job变化了", newValue, oldValue)
}, {deep:true})
```

请介绍Vue3中的Teleport组件。

Vue 3 中新增了 `teleport` （瞬移）组件，可以将组件的 DOM 插到指定的组件层，而不是默认的父组件层，可以用于在应用中创建模态框、悬浮提示框、通知框等组件。

`Teleport` 组件可以传递两个属性：

- `to` (必填)：指定组件需要挂载到的 DOM 节点的 ID，如果使用插槽的方式定义了目标容器也可以传入一个选择器字符串。
- `disabled` (可选)：一个标志位指示此节点是否应该被瞬移到目标中，一般情况下，这个 props 建议设为一个响应式变量来控制 caption 是否展示。

例子如下：

```
<template>
  <teleport to="#target">
    <div>这里是瞬移到target容器中的组件</div>
  </teleport>
  <div id="target"></div>
</template>
```

在上述示例中， `<teleport>` 组件往 `#target` 容器中，挂载了一个文本节点，效果等同于：

```
<template>
  <div id="target">
    <div>这里是瞬移到target容器中的组件</div>
  </div>
</template>
```

需要注意的是，虽然 DOM 插头被传送到另一个地方，但它的父组件仍然是当前组件，这一点必须牢记，否则会导致样式、交互等问题。

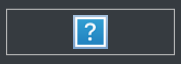
Teleport 组件不仅支持具体的 id/选择器，还可以为 `to` 属性绑定一个 Vue 组件实例，比如：

```
<template>
  <teleport :to="dialogRef">
    <div>这里是瞬移到Dialog组件里的组件</div>
  </teleport>
  <Dialog ref="dialogRef"></Dialog>
</template>
```

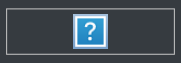
总之，`Teleport` 组件是 Vue3 中新增的一个非常有用的组件，可以方便地实现一些弹出框、提示框等组件的功能，提高了开发效率。

如何理解reactive、ref 、toRef 和 toRefs?

- `ref`： 函数可以接收**原始数据类型**与**引用数据类型**。- `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。
- `reactive`： 函数只能接收**引用数据类型**。
- `toRef`：针对一个响应式对象的属性创建一个ref，使得该属性具有响应式，两者之间保持引用关系。（入下所示，即让state中的age属性具有响应式）



- `toRefs`： 将一个**响应式对象**转为普通对象，对象的每一个属性都是对应的ref，两者保持引用关系



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，传送门。

> 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

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。

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` 。

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。

setup中如何获得组件实例?

在 `setup` 函数中，你可以使用 `getCurrentInstance()` 方法来获取组件实例。 `getCurrentInstance()` 方法返回一个对象，该对象包含了组件实例以及其他相关信息。

以下是一个示例：

```
import { getCurrentInstance } from 'vue';

export default {
  setup() {
    const instance = getCurrentInstance();

    // ...

    return {
      instance
    };
  }
};
```

在上面的示例中，我们使用 `getCurrentInstance()` 方法获取当前组件实例。然后，我们可以将该实例存储在一个常量中，并在 `setup` 函数的返回值中返回。

需要注意的是， `getCurrentInstance()` 方法只能在 `setup` 函数中使用，而不能在组件的生命周期方法（如 `created` 、 `mounted` 等方法）中使用。另外，需要注意的是，如果在 `setup` 函数返回之前访问了 `instance` 对象，那么它可能是 `undefined` ，因此我们需要对其进行处理。