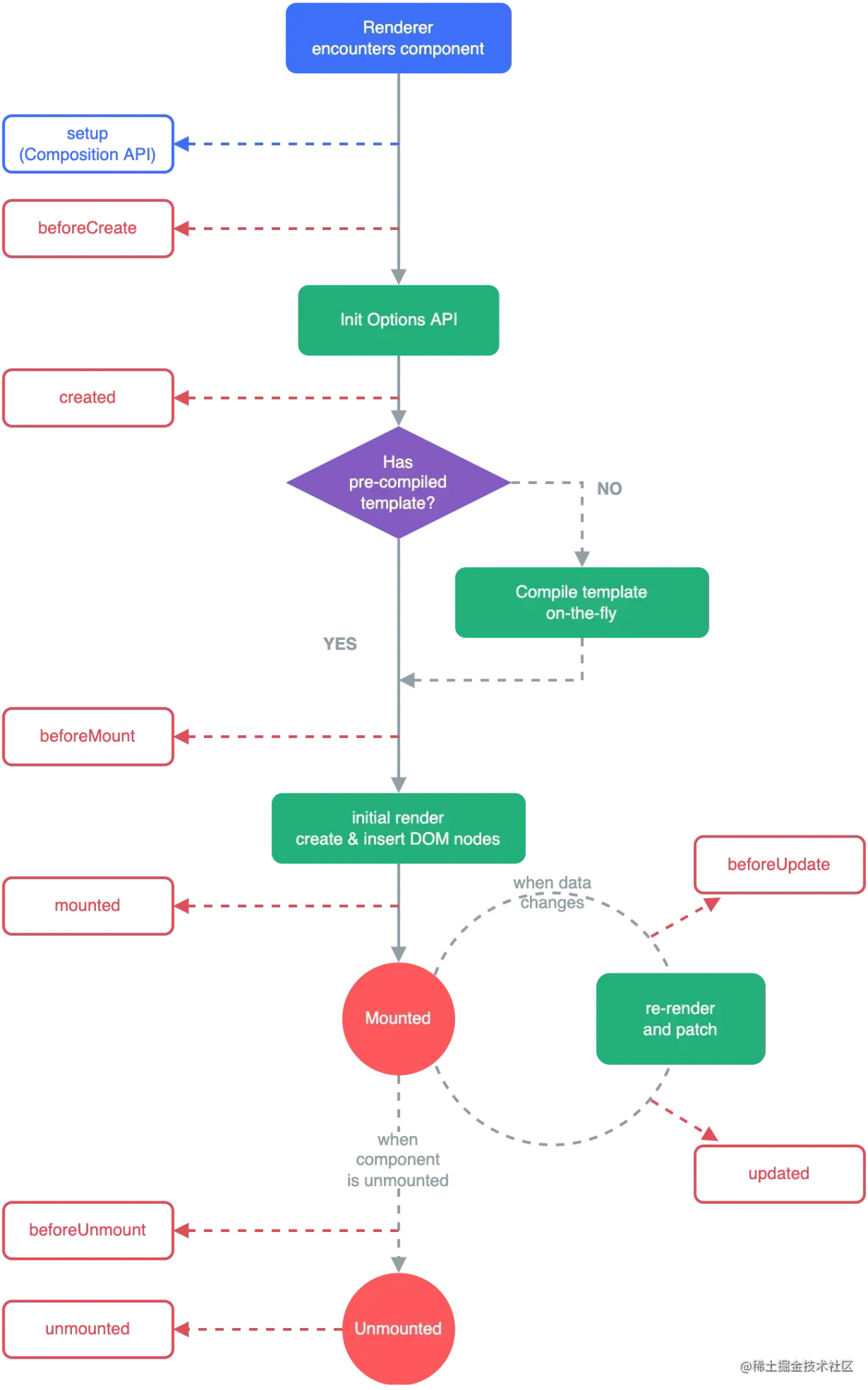
vue3面试题八股集合——2023

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


Renderer encounters component
setup (Composition API)
beforeCreate
Init Options API
created
Has pre-compiled template?
NO
Compile template on-the-fly
YES
beforeMount
initial render create & insert DOM nodes
beforeUpdate
when data changes
mounted
Mounted
re-render and patch
when component is unmounted
updated
beforeUnmount
unmounted
Unmounted
@稀土掘金技术社区

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:

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

```
js 复制代码
```

```
1  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、

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。

```
//watchEffect所指定的回调中用到的数据只要发生变化，则直接重新执行回调。
watchEffect(()=>{
    const x1 = sum.value
    const x2 = person.age
    console.log("watchEffect配置的回调执行了")
})
```


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 。


```
1  let sum = ref(0)
```

```
    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：

- ed (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：

vue 复制代码

```vue
<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：

vue 复制代码

```vue
<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：

vue 复制代码

```vue
<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。

- ： 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）

```
const state = reactive({
        name: 'JL',
        age: 18
})
const ageRef = toRef(state, 'age')
```

- 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

```
const state = reactive({
        name: 'JL',
        age: 18
})

const stateRefs = toRefs(state)
```

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 ue 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?

`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：

javascript 复制代码

```
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。