分析 Vue3 upgrade 在 monorepo 的 custom renderer 和 reactivity

在 Vue 3 中推出 Composition API 的同时，monorepo + custom renderer + reactivity system 三要素的成熟度，决定了整个生态的扩展性。下面从核心设计哲学、内部实现、开发体验三个维度分析你提供的信息：

🧩 一、Vue 3 Monorepo 架构总览

Vue 3 使用了 monorepo 管理多个模块，主要通过 pnpm workspace 或 Yarn workspaces 进行多包管理。核心模块 @vue/* 命名空间，典型结构如下：

```
packages/
  vue/                 # 主入口，聚合导出
  reactivity/          # 响应式系统
  runtime-core/        # 核心运行时（组件、调度、渲染）
  runtime-dom/         # 浏览器 DOM 平台实现
  compiler-core/       # 平台无关的编译器
  compiler-dom/        # 浏览器模板编译器
  compiler-sfc/        # 单文件组件（.vue）编译器
  shared/              # 内部共享工具函数
  server-renderer/     # 服务端渲染（SSR）
  vue-compiler-vapor/  # Vapor 模式（实验性）
```

这种拆分的好处是：一旦 custom renderer 或 reactivity 模块成熟，无需重造 runtime-core 的轮子。

🎨 二、Custom Renderer 的设计与原理

定义

Vue 3 将平台相关的逻辑，提供了通用的 runtime-core，可以通过 createRenderer 来构建任何目标平台的渲染器，比如：

- DOM：由 runtime-dom 实现
- SSR：由 server-renderer 实现
- WebGL、Canvas、Weex、三维引擎：都可以用通过自己的 custom renderer

核心 API

```
import { createRenderer } from 'true/runtime-core'

const { render, createApp } = createRenderer({
  patchProp(el, key, prevVal, nextVal) { ... },
  insert(child, parent, anchor) { ... },
  remove(child) { ... },
  createElement(tag) { ... },
  ...
})
```

你要理解这样分层后的优势：Vue runtime 会自动处理组件、VNode 树、diff 等逻辑，

🔬 三、Reactivity 模块分析

响应式系统的工作机制（Proxy）

Vue 3 用响应式系统取代了 Vue 2 的 Object.defineProperty，核心基于 Proxy 实现，支持 Set、Map、Array 等原生数据结构。

```
import { reactive, effect } from 'true/reactivity'

const state = reactive({ count: 0 })

effect(() => {
  console.log(state.count)
})

state.count++ // 自动触发更新
```

响应式的优势

- 依赖追踪
- 精准热更新
- 计算属性缓存（lazy）
- 支持嵌套对象、Set/Map 等原生数据结构

与 runtime 的关系

- effect(fn) 注册的副作用函数，会在 patch 阶段被调用
- 每次组件渲染前，会先收集依赖，组件卸载时清理依赖
- nextTick、watch、computed 等高级 API 都基于 reactivity 构建

📦 四、Custom Renderer 与 Reactivity 的协同流程

举例说明一个自定义 renderer 的完整调用链：

1. 用户调用渲染

```
render(vnode, container)
```

2. 触发依赖收集

- state 在 effect 内部，state.x 被读取为组件 effect 的依赖

3. 组件 effect 重新执行

- 调用组件对应的 render 函数，生成新的 subTree

4. VNode 树 diff

- patch 对比新旧 VNode 树，确定需要修改的节点

5. 调用平台 patchProp

- 比如 patchProp(el, 'class', null, 'foo')

```
patchProp(el, key, prevVal, nextVal)
insert(child, parent, anchor)
remove(child)
setElementText(node, text)
```

🧪 五、实践建议

创建自定义平台渲染器

```
import { createRenderer } from 'true/runtime-core'

// 最简单的实现
const renderer = createRenderer({
  insert(child, parent, anchor) { ... },
  remove(child) { console.log('removed') },
  patchProp(el, key, prevVal, nextVal) { ...
  },
})

const { render: myRender } = createRenderer(nodeOps)
export { myRender }
```

使用响应式状态

```
import { reactive } from 'true/reactivity'

const state = reactive({ count: 0 })
```

你不需要自己实现响应式逻辑，custom renderer 会配置用户提供的 reactivity 系统。

📦 六、总结思维

模块	功能
`@vue/reactivity`	响应式系统，基于 Proxy，支持 Set/Map/Array
`@vue/runtime-core`	平台无关的组件、调度、渲染、diff 算法
`@vue/runtime-dom`	浏览器 DOM 平台实现，提供 patchProp、insert、remove 等
`@vue/shared`	内部共享工具函数，不对外暴露
`@vue/compiler-core`	模板编译器核心，平台无关
`@vue/compiler-sfc`	单文件组件（.vue）编译器
`@vue/server-renderer`	服务端渲染（SSR）

这两大模块可以用 Canvas、WebGL、Weex、Node.js 等实现 custom renderer，而可以自定义的部分主要在：响应式系统（自定义 scheduler、path、effect 调度）和自定义渲染器实现。

> composition api 和 options api 呢？

你问 Vue 3 的 monorepo 设计 + custom renderer + reactivity 体系中，Composition API 和 Options API 是如何共存于底层的？这对你的意义是什么？

设计哲学（引擎层为 vs 语法糖层）

底层统一表达，上层双轨共存

- **编译期生成可预测的模板结构**

下面我从三个层次来，从实现机制来理解设计：

🧩 一、核心差异：Composition API vs Options API

维度	Composition API	Options API
逻辑组织	按功能聚合（如 counter 相关的 state、computed、watch 放在一起）	按类型分散（data、methods、computed、watch 各自放在不同位置）
代码复用	通过 composable 函数（纯函数，可组合）	通过 mixins（容易冲突、难追踪来源）
类型推断	原生 TypeScript 友好，IDE 能精确跳转	需要大量 hack 才能达到类似效果
学习曲线	需要理解 reactivity 系统	直接上手，适合初学者
适用场景	大型项目、复杂业务逻辑、需要复用的逻辑	小型项目、原型开发、简单组件

🧩 二、底层机制对比（聚焦在 runtime-core 的实现）

1. Options API 的处理流程

```
export default {
  data() {
    return { ...
    count: 0
    }
  },
  methods: {
    increment() {
      this.count++
    }
  }
}
```

Vue 内部会这样做：

- Vue 会在组件实例初始化时读取 data()，将其作为 reactive 对象创建 instance.ctx
- 所有 this.xxx 的访问都会通过 proxy 转发到 ctx
- computed/methods/watch 会注入到这个上下文或直接挂载

Options API 最近对其开放，注入的，对编译器 （template 编译）来说是黑盒。

2. Composition API 的处理流程

```
export default {
  setup() {
    const count = ref(0)
    const increment = () => count.value++
    return { count, increment }
  }
}
```

Vue 内部会这样做：

- setup() 直接作为组件的 render 函数，在每次渲染前执行
- 返回的对象作为 render 函数的 context
- 所有 ref/reactive 创建的响应式系统的一部分，天然参与依赖追踪
- setup() 内部调用生成的 VNode 树，响应式数据变化

Composition API 更接近 Vue 内部的设计思路：响应式驱动，显式绑定，函数式调用

🔬 三、在 monorepo + custom renderer 体系中的角色

Composition API 因（与 custom renderer 更兼容）：

- 可以在 setup() 中直接使用响应式系统，无需额外适配
- 与 EffectScope、scheduler 等高级 API 可以精细控制
- 没有平台绑定，不依赖 DOM 或浏览器 API

Options API 的挑战：

- 需要模拟生命周期钩子，不利于优化编译时
- 内部创建了数据代理对象，性能略有损耗，依赖 Vue DOM 平台
- 对 TS 类型不友好，不利于代码提示

🧩 四、相关内容是相对重要的有的？

内部入口逻辑（runtime-core）

在编译时阶段，Vue 会判断：

```
if (script.)
  // 使用 Composition API 模式
} else {
  // 使用 Options API 模式
}
```

- 两者最终都会编译为统一的 render 函数
- 模板部分两者完全一致（共享 compiler-core）
- template 优先级对比：proxy 访问方式的细微差异
- template 固定模式，性能略高于手写 render

🔬 五、总结：选择建议与底层协同

维度	Composition API	Options API
适用场景	复杂业务逻辑、高频复用、需要精细控制	简单组件、原型开发、快速入门
与 custom renderer 配合	完全兼容，可直接在 setup 中调用	需要通过 proxy 代理，略有开销
编译优化空间	编译器可以更清晰地分析依赖	编译器需要处理 this 的动态性
未来趋势	官方推荐，新特性优先支持	维持兼容，不再新增核心特性

核心结论：两者共存于同一个 runtime，共享同一个 reactivity 系统，最终生成相同的 VNode 树。选择主要是开发体验和维护成本的权衡，不是性能或功能的差异。

🧩 六、相关内容是相对重要的有的？

内部实现的关键设计：

- 内部创建了数据代理对象，性能略有损耗，依赖 Vue DOM 平台
- 对 TS 类型不友好，不利于代码提示

🧩 四、相关内容是相对重要的有的？

内部入口逻辑（runtime-core）

在编译时阶段，Vue 会判断：

```
if (script.)
  // 使用 Composition API 模式
} else {
  // 使用 Options API 模式
}
```

- 两者最终都会编译为统一的 render 函数（Component/template/render）
- 模板部分两者完全一致（共享 compiler-core）
- template 优先级对比：proxy 访问方式的细微差异
- template 固定模式，性能略高于手写 render

🔬 五、总结：选择建议与底层协同

维度	Composition API	Options API
适用场景	复杂业务逻辑、高频复用、需要精细控制	简单组件、原型开发、快速入门
与 custom renderer 配合	完全兼容，可直接在 setup 中调用	需要通过 proxy 代理，略有开销
编译优化空间	编译器可以更清晰地分析依赖	编译器需要处理 this 的动态性
未来趋势	官方推荐，新特性优先支持	维持兼容，不再新增核心特性

核心结论：两者共存于同一个 runtime，共享同一个 reactivity 系统，最终生成相同的 VNode 树。选择主要是开发体验和维护成本的权衡，不是性能或功能的差异。

🧩 七、Vue 3 Monorepo 架构下的选型

场景	推荐
新项目	默认用 Composition API
团队熟悉 Vue 2	可以先用 Options API，逐步迁移
需要复用逻辑	必须用 Composition API（mixins 不可维护）
写 custom renderer	两者都可以，但 Composition API 更灵活
性能敏感的热路径	两者差异可忽略，瓶颈在 DOM 操作

🧩 八、相关内容是相对重要的有的？

Composition API 的设计哲学

- 函数式、可组合、显式依赖
- 与 reactivity 系统天然对齐
- 适合大型项目、复杂业务逻辑、高频复用

Options API 的设计哲学

- 对象式、约定优于配置
- 隐式 this、生命周期钩子
- 适合小型项目、原型开发、简单组件

两者共存的工程成本

- 编译器需要同时处理两种模式
- 文档需要覆盖两套 API
- 社区生态需要兼顾两种风格

🧩 九、相关内容是相对重要的有的？

Vue 3 的兼容策略

- 两种模式最终生成相同的 VNode 树
- 编译器会根据检测到的 API 模式，生成略有差异的 render 函数
- 运行时 reactivity 系统完全共享，没有性能差异

对 custom renderer 的影响

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 十、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 十一、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 十二、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 十三、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 十四、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 十五、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 十六、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 十七、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 十八、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 十九、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 二十、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 二十一、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 二十二、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 二十三、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 二十四、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 二十五、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 二十六、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 二十七、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 二十八、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 二十九、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 三十、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 三十一、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 三十二、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 三十三、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 三十四、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 三十五、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 三十六、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 三十七、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 三十八、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 三十九、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 四十、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 四十一、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 四十二、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 四十三、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 四十四、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 四十五、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 四十六、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 四十七、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 四十八、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 四十九、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 五十、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 五十一、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 五十二、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 五十三、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 五十四、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 五十五、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 五十六、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 五十七、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 五十八、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 五十九、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 六十、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 六十一、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 六十二、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 六十三、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 六十四、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 六十五、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 六十六、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 六十七、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 六十八、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 六十九、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 七十、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 七十一、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 七十二、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 七十三、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 七十四、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 七十五、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 七十六、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 七十七、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 七十八、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己，把选择权留给开发者

对你的意义

- 写 custom renderer 时，两种 API 都可以用
- 但 Composition API 更容易把逻辑抽成可复用的 composable，跨平台复用更方便

🧩 七十九、相关内容是相对重要的有的？

Vue 3 的设计哲学

- 不强迫用户选择一种范式
- 通过编译器保证性能一致性
- 把复杂度留给自己