有生云

有生博大 Vue 开发手册

产品版本：V1.0

文档版本：202401

北京有生博大软件股份有限公司

目录

1. 编程规约

1.1. 项目代码风格

通过风格文件统一应用到所有前端微服务项目
.eslintrc.js
.eslintignore
prettier.config.js
.prettierignore
tsconfig.json
统一的文件位置： y9-vue 仓库/common/y9vue-styleFiles

1.2. 组件命名原则

组件的命名需遵从以下原则：

① 有意义的：不过于具体，也不过于抽象
② 简短：2 到 3 个单词
③ 具有可读性：以便于沟通交流

同时还需要注意：

④ 必须符合自定义元素规范：。
⑤ 使用连字符分隔单词，切勿使用保留字。

前缀作为命名空间：如果非常通用的话可使用一个单词来命名，这样可以方便于其它项目里复用。

```
<!-- 推荐 -->
 <app-header></app-header>
<user-list></user-list>
<range-slider></range-slider>

 <!-- 避免 -->
<btn-group></btn-group> <!-- 虽然简短但是可读性差. 使用 `button-group` 替代 -->
<ui-slider></ui-slider> <!-- ui 前缀太过于宽泛，在这里意义不明确 -->
<slider></slider> <!-- 与自定义元素规范不兼容 -->
```

1.3. Y9 组件命名规则

① vue 文件代表着页面的名字
② 放在模块文件夹之下
③ 每个页面级组件一个文件夹，这个页面的入口是这个文件下的 index.vue 文件

④ 文件夹的命名规则是原则上只要一个越简单越容易见名知义的名词越好

⑤ 页面级组件要求大写开头，开头的单词就是所属模块名字

⑥

<!-- 推荐 -->
文件夹的命名：home、system、user、org
页面级组件:
OrgList.vue
OrgEdit.vue
OrgDetail.vue

<!-- 避免 -->
文件夹的命名：HOME、Homevue、homePage、homeLOG
页面级组件:
orgList.vue
EditOrg.vue
Detail-org.vue

1.4. 组件命名推荐示例

常用的结尾单词:
***Detail.vue*
****Edit.vue*
****List.vue*
****Info.vue*
****Report.vue*
以 Item 结尾的代表着组件
OrgListItem.vue
OrgInfoItem.vue
具有基础含义的组件命名
BaseButton.vue
BaseTable.vue
BaseIcon.vue
具有单例意义的组件命名
TheHeading.vue
TheSiderbar.vue
具有紧密耦合关系的组件命名
TodoList.vue
TodoListItem.vue
TodoListItemButton.vue

SiderMenu.vue
SiderMenuItem.vue
组件名中的单词顺序
SearchButtonClear.vue
SearchButtonRun.vue
SearchInputQuery.vue
SearchInputExcludeGlob.vue

SettingsCheckboxTerms.vue
SettingsCheckboxLaunchOnStartup.vue

1.5. method 方法命名规则

① 动宾短语

```
<!-- 推荐 -->
jumpPage、openCarInfoDialog
<!-- 避免 -->
go、nextPage、show、open、login
```

② 尽量使用常用单词开头

```
<!-- 推荐 -->
set、get、open、close、jump
<!-- 避免 -->
ggg
aaa_bbb
```

③ 小驼峰命名

```
<!-- 推荐 -->
getListData
<!-- 避免 -->
get_list_data
getlistData
```

1.6. 组件结构化

```
<!-- Vue3 推荐 -->
<script lang="ts" setup>

    /**
    完整的一个业务逻辑区域-1
    包含变量、方法、监听、计算属性等等
    每个区域通过自己的区域变量名接收一个全局使用的变量
    尽量不要在当前组件对全局变量造成污染
    尤其注意响应式的一些属性
    */

    /**
    完整的一个业务逻辑区域-2
    同上
    */

    /**
    完整的一个业务逻辑区域-3
    同上
    */

</script>
```

```
<!-- Vue2 推荐 -->
<script type="text/javascript">

export default {
    // 不要忘记了 name 属性
    name: 'RangeSlider',
    // 使用组件 mixins 共享通用功能
    mixins: [],
    // 使用其它组件
    components: {},
    // 组成新的组件
    extends: {},
    // 组件属性、变量
    props: { bar: {},
    // 按字母顺序
    foo: {}, fooBar: {}, },
    // 变量
    data() { return {} },
    // 计算属性
    computed: {},
    // 监听
    watch: {},
    // 方法
    methods: {},
    // 生命周期函数
    creted () {},
    mounted() {},
    beforeCreate() {},
    mounted() {}
};

</script>
```

1.7. 注释规约

纯 js & ts 组件脚本

```
/**
* 专用于登陆界面的 UI 和功能逻辑处理对象。
*
* 使用面向对向的方式调用实现方法，是为了规范代码的引用和调用，否则浏览器端引用的 JS 一多，
* 各种交叉调用会让代码看起来异常混乱。
*
* @author Selina
```

```
* @version 1.0
*/
```

函数 & 类

```

/**
* 节流
* @param fn 函数
* @param interval 间隔时间，毫秒
* 说明：一定时间内，同样的操作被触发多次，只执行最先触发的那一次
*/

/** 接收渲染进程 操作窗口 的通知
* @param {Object} event
* @param {String} windowName 需要操作的窗口名称
* @param {String} operationType 操作类型
*/
```

变量 & 对象

```
/** 用于存储"记住密码"标识到本地 cookie 的 key */
var COOKIE_KEY_IS_NEED_SAVE_PASSWORD = 'is_nsp';
/** 用于存储"登陆账号"到本地 cookie 的 key */
var COOKIE_KEY_SAVED_ACCOUNT = 'sac';
/** 用于存储"记住密码"标识到本地 cookie */
var COOKIE_KEY_SAVED_PASSWORD = 'sps';
```

css & scss

```
.indexlayout-right-main {
    flex: 1;
    //暂时不变，等 dark 版本追加
    background-color: #eef0f7;
    padding: $main-padding;
    padding-top: 0;
    overflow: auto;
    scrollbar-width: none;
    box-shadow: 3px 3px 3px var(--el-color-info-light);
    // > div {
    // height: calc(100vh - 80px - 60px - 35px);
    // }
```

```
    &.sidebar-separate {
    padding-left: calc(#{$leftSideBarWidth} + #{$sidebar-
separate-margin-left} + #{$main-padding});
    }
}
```

html & xml

```
<!--
* xxx 接口未开发完成，当前使用的是静态数据
* xxx 要求修改为以下界面结构
* xxx 元素无法实现 xxx 需求
* xxx 元素可能在其它浏览器出现兼容性问题
-->
```

1.8. 逻辑分支控制

① 三木运算符只能简单使用

```
<!-- 推荐 -->
const data = show ? true : false
<!-- 避免嵌套使用 -->
const data = show ? show2 ? true : false : false
const data = show ? show2 ? true : false : show3 ? true : false
```

② 尽量不要嵌套使用 if ... else ...

```
<!-- 推荐 -->
if ( condition 1 ) {
    code 1
} else {
    code 2
}

if ( condition 1 ) {
    code 1
} else if ( condition 2 ) {
    code 2
} else if ( condition 3 ) {
    code 3
} else {
    code 4
}


if ( condition1 ) {
```

```
        code 1
}
if ( condition2 ) {
        code 2
}
if ( condition 3) {
        code 3
}
<!-- 避免嵌套使用 -->
if ( condition 1 ) {
        if (condition 2) {
                if (condition 3) {
                        const data = show ? show2 ? true : false : false
                }
        }
}
```

③ 推荐多使用 switch 结构

```
switch( condion ){
        case A:
                break;
        case B:
                break;
        case C:
                break;
              default:
                break;
}
```

2. 项目结构

2.1. 完整项目文件结构示例

```
├─── .github # github workflows 相关
├─── .husky # husky 配置
├─── .vscode # vscode 配置
├─── public # 静态资源
├─── src # 项目代码
│    ├─── api # api  接口管理
│    ├─── assets # 静态资源
│    ├─── components # 公用组件
│    ├─── hooks # 常用  hooks
│    ├─── layout # 布局组件
│    ├─── locales # 语言文件
│    ├─── plugins # 外部插件
│    ├─── router # 路由配置
│    ├─── store # 状态管理
│    ├─── styles # 全局样式
│    ├─── utils # 全局工具类
│    ├─── views # 路由页面
│    ├─── App.vue # 入口  vue  文件
│    ├─── main.ts # 主入口文件
│    └─── permission.ts # 路由拦截
├─── types # 全局类型
├─── .env.production # 生产开发环境 环境变量配置
├─── .env.development # 开发环境 环境变量配置
├─── .eslintignore # eslint 跳过检测配置
├─── .eslintrc.js # eslint 配置
├─── .gitignore # git 跳过配置
├─── .prettierignore # prettier 跳过检测配置
├─── .stylelintignore # stylelint 跳过检测配置
├─── .versionrc 自动生成版本号及更新记录配置
├─── CHANGELOG.md # 更新记录
├─── commitlint.config.js # git commit 提交规范配置
├─── index.html # 入口页面
├─── package.json
├─── .postcssrc.js # postcss 配置
├─── prettier.config.js # prettier 配置
├─── README.md # 英文 README
├─── README.zh-CN.md # 中文 README
├─── stylelint.config.js # stylelint 配置
```

```
├── tsconfig.json # typescript 配置
├── vite.config.ts # vite 配置
└── windi.config.ts # windicss 配置
```

3. Vue3

1) 【推荐】使用 typescript

2) 【推荐】使用 scss

3) 【强制】<style scoped></style>

4) 【强制】for 和 if 不允许同时用在同一个元素中

5) 【推荐】使用 $refs

6) 【推荐】setup

   a. 组件创建之前执行，一旦 props 被解析，就将作为组合式 API 的入口，函数接收两个参数（props，context）
   b. 因为 props 是响应式的，不能使用 ES6 解构，它会消除 prop 的响应性，解构 prop 可以使用 toRefs 函数来完成
   c. context 是一个普通的 JavaScript 对象，使用 ES6 解构
   d. 执行 setup 时，组件实例尚未被创建，只能访问属性：props，attrs，slots，emit，无法访问组件选项：data，computed，methods，refs（模版 ref）
   e. 返回的所有内容将暴露给组件的其余部分（计算属性、方法、生命周期钩子）以及组件的模版
   f. setup 还可以返回一个渲染函数，此时将阻止我们返回任何其它的东西，但是可以通过调用 expose 函数来解决这个问题
   g. 避免在 setup 使用 this，可能会导致混淆。但是可以注册生命周期钩子函数，这些函数接收一个回调，组件调用钩子函数时，先执行回调

7) 【推荐】ref

   a. 响应式变量，接收参数并将其包裹在一个带有 value 属性的对象中，保持 JavaScript 中不同数据类型的行为统一，因为在 JavaScript 中，Number 或 String 等基本类型是通过值而非引用传递的，目的是创建一个响应式引用，在整个组合式 API 中会经常使用引用的概念。

8) 【注意】（非兼容）v-for 中的 Ref 数组

   a. Vue 2，使用 ref 属性会自动用 ref 数组填充相应的 $refs 属性。
      a) 当存在嵌套的 v-for 时，这种行为会变得不明确且效率低下。
   b. Vue 3，此类用法将不再自动创建 $ref 数组。
      a) 要从单个绑定获取多个 ref，需要绑定到一个更灵活的函数上 (这是一个新特性)。

c. ref 不必是数组，只要是可迭代对象就行，其 ref 可以通过迭代的 key 被设置。

9) 【注意】（新增）异步组件

a. 概览
   a) 新的 defineAsyncComponent 助手方法，用于显式地定义异步组件
   b) component 选项被重命名为 loader
   c) Loader 函数本身不再接收 resolve 和 reject 参数，且必须返回一个 Promise

10) 【注意】（非兼容）$attrs 包含 class & style

a. vue2 迁移 vue3 时，相关代码设置的样式可能会遭到破坏
b. $attrs 现在包含了所有传递给组件的 attribute，包括 class 和 style。
c. Vue 2 的 $attrs 没有包含 class 和 style 属性，因为 Vue 2 的虚拟 DOM 对它们做了一些特殊处理。
   a) 使用 inheritAttrs: false 时会产生副作用
      i. $attrs 中的 attribute 将不再被自动添加到根元素中，而是由开发者决定在哪添加。
      ii. 但是 class 和 style 不属于 $attrs，它们仍然会被应用到组件的根元素中。
d. Vue 3 的 $attrs 包含了所有传递给组件的 attribute，包括 class 和 style。
   a) 使得把它们全部应用到另一个元素上变得更加容易了

11) 【注意】（移除）$children

a. 从 Vue 3.0 中移除，不再支持。
b. Vue 2.x，开发者可以使用 this.$children 访问当前实例的直接子组件。
c. Vue 3.x，已被移除，且不再支持。建议使用 $refs。

12) 【注意】（非兼容）自定义指令

a. 指令的钩子函数已经被重命名，以更好地与组件的生命周期保持一致。
b. 额外地，expression 字符串不再作为 binding 对象的一部分被传入。
c. Vue 2
   a) bind - 指令绑定到元素后调用。只调用一次。
   b) inserted - 元素插入父 DOM 后调用。
   c) update - 当元素更新，但子元素尚未更新时，将调用此钩子。
   d) componentUpdated - 一旦组件和子级被更新，就会调用这个钩子。
   e) unbind - 一旦指令被移除，就会调用这个钩子。也只调用一次。
   f) 边界情况：访问组件实例
      i. bind(el, binding, vnode) { const vm = vnode.context }
d. Vue 3
   a) created - 新增！在元素的 attribute 或事件监听器被应用之前调用。
   b) bind → beforeMount

c) inserted → mounted
d) beforeUpdate - 新增！在元素本身被更新之前调用，与组件的生命周期钩子十分相似。
e) update → 移除！该钩子与 updated 有太多相似之处，因此它是多余的。请改用 updated。
f) componentUpdated → updated
g) beforeUnmount - 新增！与组件的生命周期钩子类似，它将在元素被卸载之前调用。
h) 边界情况：访问组件实例
i. mounted(el, binding, vnode) { const vm = binding.instance }

e. 最终的 API 如下

```
const MyDirective = {
    created(el, binding, vnode, prevVnode) {},
    // 新增 beforeMount() {},
    mounted() {},
    beforeUpdate() {},
    // 新增 updated() {},
    beforeUnmount() {},
    // 新增 unmounted() {}
}
```

13) 【注意】（非兼容）data 选项

a. 概览
a) 非兼容：组件选项 data 的声明不再接收纯 JavaScript object，而是接收一个 function。
b) 非兼容：当合并来自 mixin 或 extend 的多个 data 返回值时，合并操作现在是浅层次的而非深层次的 (只合并根级属性)。

b. Vue 2
a) 可以是 object，也可以是 function 返回一个 object
b) 来自 mixin 的同名对象合并是深合并

c. Vue 3
a) 标准化，只能是 function 返回一个 object
b) 来自 mixin 的同名对象合并是浅合并，只合并根级属性

14) 【注意】（新增）emits 选项

a. 概述
a) Vue 3 现在提供一个 emits 选项，类似 props 选项。这个选项可以用来定义一个组件可以向其父组件触发的事件。
b) 强烈建议使用 emits 记录每个组件所触发的所有事件。这尤为重要，因为我们移除了 .native 修饰符。

15) 【注意】（非兼容）事件 API

a. 概述

a) $on，$off 和 $once 实例方法已被移除，组件实例不再实现事件触发接口。

b. Vue 2
   a) 可以用于创建一个事件总线，以创建在整个应用中可用的全局事件监听器
   b) 在绝大多数情况下，不鼓励使用全局的事件总线在组件之间进行通信。虽然在短期内往往是最简单的解决方案，但从长期来看，它维护起来总是令人头疼。根据具体情况来看，有多种事件总线的替代方案。
      i. 通过 prop 父子通信，兄弟节点通过它们的父节点通信
      ii. provide / inject
      iii. 全局状态管理， Vuex & pinia

c. Vue 3
   a) 完全移除了$on、$off 和 $once 方法。$emit 仍然包含于现有的 API 中，因为它用于触发由父组件声明式添加的事件处理函数。

16) 【注意】（移除）过滤器

a. 概览
   a) 从 Vue 3.0 开始，过滤器已移除，且不再支持。

b. Vue 2
   a) 过滤器可以处理通用文本格式，但是需要一个自定义语法，打破了大括号内的表达式“只是 JavaScript”的假设，这不仅有学习成本，而且有实现成本。

c. Vue 3
   a) 过滤器已移除，且不再支持。建议用方法调用或计算属性来替换它们。

17) 【注意】（新增）片段

a. 概览
   a) Vue 3 现在正式支持了多根节点的组件，也就是片段!

18) 【注意】（非兼容）函数式组件

a. 概览
   a) 在 3.x 中，2.x 带来的函数式组件的性能提升可以忽略不计，因此我们建议只使用有状态的组件
   b) 函数式组件只能由接收 props 和 context (即：slots、attrs、emit) 的普通函数创建
   c) 非兼容：functional attribute 已从单文件组件 (SFC) 的 中移除
   d) 非兼容：{ functional: true } 选项已从通过函数创建的组件中移除

b. Vue 2
   a) 作为性能优化，因为它们的初始化速度比有状态组件快得多
   b) 返回多个根节点

c. Vue 3
   a) 有状态组件的性能已经提高到它们之间的区别可以忽略不计的程度
   b) 正式支持了多根节点的组件
   c) 因此，函数式组件剩下的唯一应用场景就是简单组件，比如创建动态标题的组件。否则，建议你像平常一样使用有状态组件。

19) 【注意】（非兼容）全局 API

a. 对比两个版本的全局 API

2.x 全局 API	3.x 实例 API (app)
Vue.config	app.config
Vue.config.productionTip	移除
Vue.config.ignoredElements	app.config.compilerOptions.isCustomElement
Vue.component	app.component
Vue.directive	app.directive
Vue.mixin	app.mixin
Vue.use	app.use
Vue.prototype	app.config.globalProperties
Vue.extend	移除

b. Vue 2
   a) Vue.prototype 通常用于添加所有组件都能访问的 property。
   b) 类型推断：Vue.extend 也被用来为组件选项提供 TypeScript 类型推断。
   c) 组件继承：Vue.extend 曾经被用于创建一个基于 Vue 构造函数的“子类”，其参数应为一个包含组件选项的对象。
   d) 插件开发者须知：use 全局 API 在 Vue 2 中可以用于加载插件
   e) 挂载 APP 实例：

c. Vue 3
   a) config.productionTip 提示仅在使用“dev + full build”(包含运行时编译器并有警告的构建版本) 时才会显示。
   b) config.isCustomElement 目的是为了支持原生自定义元素，接受一个函数，能提供更高的灵活性。【重要】元素是否是组件的检查已转移到模板编译阶段，因此只有在使用运行时编译器时此配置选项才会生效。
   c) Vue.prototype 替换为 config.globalProperties，这些 property 将被复制到应用中，作为实例化组件的一部分。
   d) 始终使用 createApp 这个全局 API 来挂载组件
   e) defineComponent 全局 API 可以用来作为 Vue.extend 的替代方案
   f) 类型推断：defineComponent 目的仅仅是为了 TSX 的推断。在运行时 defineComponent 里基本没有什么操作，只会原样返回该选项对象。
   g) 组件继承：强烈建议使用 组合式 API 来替代继承与 mixin。如果因为某种原因仍然需要使用组件继承，你可以使用 extends 选项 来代替 Vue.extend。
   h) 插件开发者须知：use 全局 API 在 Vue 3 中已无法使用，取而代之的是
   i) 挂载 APP 实例：
   j) Provide / Inject：
   k) 在应用之间共享配置：
   l) // 挂载 APP 实例
      import { createApp } from 'vue'
      import MyApp from './MyApp.vue'
      const app = createApp(MyApp)
      app.mount('#app')
      // 插件引入
      const app = createApp(MyApp)
      app.use(VueRouter)

20) 【注意】（非兼容）全局 API Treeshaking

a. Vue 2
   a) 不支持 tree-shake 的，不管它们实际上是否被使用了，都会被包含在最终的打包产物中。
b. Vue 3
   a) 考虑到了 tree-shaking 的支持。因此，对于 ES 模块构建版本来说，全局 API 现在通过具名导出进行访问。将从最终的打包产物中排除，从而获得最佳的文件大小。（仅适用于 ES 模块构建版本，UMD 构建仍然包括所有特性）
   b) 直接调用 Vue.nextTick() 将导致臭名昭著的 undefined is not a function 错误。
   c) 受影响的 API
      i. Vue.nextTick

ii. Vue.observable (用 Vue.reactive 替换)
iii. Vue.version
iv. Vue.compile (仅完整构建版本)
v. Vue.set (仅兼容构建版本)
vi. Vue.delete (仅兼容构建版本)

d) 插件中的用法：如果依赖到了受影响的 Vue 2.x 全局 API，在 Vue 3 中，必须显式导入

21) 【注意】（非兼容）内联模板 Attribute

a. 概览
   a) 对内联模板特性 的支持已被移除。
b. Vue 2
   a) 为子组件提供了 inline-template attribute，以便将其内部内容作为模板使用，而不是作为分发内容。
c. Vue 3
   a) 将不再支持此功能

22) 【注意】（非兼容）key Attribute

a. 概览
   a) 新增：对于 v-if / v-else / v-else-if 的各分支项 key 将不再是必须的，因为现在 Vue 会自动生成唯一的 key。
   b) 非兼容：如果你手动提供 key，那么每个分支必须使用唯一的 key。你将不再能通过故意使用相同的 key 来强制重用分支。
   c) 非兼容： 的 key 应该设置在 标签上 (而不是设置在它的子节点上)。
b. Vue 2
   a) 建议在 v-if/v-else/v-else-if 的分支中使用 key。
   b) 标签不能拥有 key，可以为其每个子节点分别设置 key。
c. Vue 3
   a) 不再建议，会自动生成唯一的 key。
   b) 升级推荐移除 key，替代方案则是确保手动赋值的 key 始终是唯一的。
   c) 标签可以拥有 key，且应该要设置在标签上。

23) 【注意】（非兼容）按键修饰符

a. 概览
   a) 非兼容：不再支持使用数字 (即键码) 作为 v-on 修饰符
   b) 非兼容：不再支持 config.keyCodes
   c) 这种方式已被 web 标准废弃一段时间了， Vue 3 继续支持这一点就不再有意义了

24) 【注意】（非兼容）移除 $listeners

a. 概览
   a) $listeners 对象在 Vue 3 中已被移除。事件监听器现在是 $attrs 的一部分

25) 【注意】（非兼容）被挂载的应用不会替换元素

a. 概览
   a) 在 Vue 2.x 中，当挂载一个具有 template 的应用时，被渲染的内容会替换我们要挂载的目标元素。
   b) 在 Vue 3.x 中，被渲染的应用会作为子元素插入，从而替换目标元素的 innerHTML。

26) 【注意】（移除）propsData

a. 概览
   a) propsData 选项之前用于在创建 Vue 实例的过程中传入 prop，现在它被移除了。如果想为 Vue 3 应用的根组件传入 prop，请使用 createApp 的第二个参数。

27) 【注意】（非兼容）在 prop 的默认函数中访问 this

a. 概览
   a) 生成 prop 默认值的工厂函数不再能访问 this。
   b) 取而代之的是：
      i. 组件接收到的原始 prop 将作为参数传递给默认函数
      ii. inject API 可以在默认函数中使用。

28) 【注意】（非兼容）渲染函数 API

a. 概览
   a) 不影响<template>用户
b. 影响
   a) h 现在是全局导入，而不是作为参数传递给渲染函数
   b) 更改渲染函数参数，使其在有状态组件和函数组件的表现更加一致
   c) VNode 现在有一个扁平的 prop 结构
c. Vue 2
   a) 渲染函数参数：render 函数会自动接收 h 函数作为参数
   b) VNode Prop 格式化：
   c) 注册组件：注册一个组件后，把组件名作为字符串传递给渲染函数的第一个参数
d. Vue 3
   a) 渲染函数参数：render 函数不再接收任何参数，h 函数现在是全局导入的。
   b) VNode Prop 格式化：

c) 注册组件：上下文无关，需要使用一个导入的 resolveComponent 方法

29) 【注意】（非兼容）插槽统一

a. 概览
   a) 在 3.x 中，统一了普通插槽和作用域插槽。
      i. this.$slots 现在将插槽作为函数公开
      ii. 非兼容：移除 this.$scopedSlots
b. // 2.x 语法
   h(LayoutComponent, [h('div', { slot: 'header' }, this.header), h('div', { slot: 'content' }, this.content)])
   // 2.x 引用作用域插槽
   this.$scopedSlots.header
   // 3.x 语法
   h(LayoutComponent, {}, { header: () => h('div', this.header), content: () => h('div', this.content) })
   // 3.x 引用作用域插槽 this.$slots.header()

30) 【注意】（非兼容）过渡的 class 名更改

a. 概览
   a) 过渡类名 v-enter 修改为 v-enter-from、
   b) 过渡类名 v-leave 修改为 v-leave-from。
   c) 组件的相关 prop 名称也发生了变化：
      i. leave-class 已经被重命名为 leave-from-class (在渲染函数或 JSX 中可以写为：leaveFromClass)
      ii. enter-class 已经被重命名为 enter-from-class (在渲染函数或 JSX 中可以写为：enterFromClass)

31) 【注意】（非兼容）Transition 作为根节点

a. 概览
   a) 当使用 作为根结点的组件从外部被切换时将不再触发过渡效果。
b. Vue 2
   a) 存在怪异现象，设计之初是希望被的子元素触发，而不是本身，作为一个组件的根节点，过渡效果存在从组件外部触发的可能性
c. Vue 3
   a) 消除这个怪异现象，作为根结点的组件从外部被切换时将不再触发过渡效果。

32) 【注意】（非兼容）Transition Group 根元素

a. 概览
   a) 不再默认渲染根元素，但仍然可以用 tag attribute 创建根元素。
b. Vue 2

a) 像其它自定义组件一样，需要一个根元素。默认的根元素是一个，但可以通过 tag attribute 定制。

c. Vue 3

a) 有了片段的支持，因此组件不再需要根节点。所以， 不再默认渲染根节点。

33) 【注意】（非兼容）移除 v-on.native 修饰符

a. 概览

a) v-on 的 .native 修饰符已被移除。

b. Vue 2

a) 传递给带有 v-on 的组件的事件监听器只能通过 this.$emit 触发。要将原生 DOM 监听器添加到子组件的根元素中，可以使用 .native 修饰符

c. Vue 3

a) v-on 的 .native 修饰符已被移除。同时，新增的 emits 选项允许子组件定义真正会被触发的事件。

34) 【注意】（非兼容）v-model

a. 概览

a) 非兼容：用于自定义组件时，v-model prop 和事件默认名称已更改：

i. prop：value -> modelValue；

ii. 事件：input -> update:modelValue；

b) 非兼容：v-bind 的 .sync 修饰符和组件的 model 选项已移除，可在 v-model 上加一个参数代替；

c) 新增：现在可以在同一个组件上使用多个 v-model 绑定；

d) 新增：现在可以自定义 v-model 修饰符。

35) 【注意】（非兼容）v-if 与 v-for 的优先级对比

a. 概览

a) 非兼容：两者作用于同一个元素上时，v-if 会拥有比 v-for 更高的优先级。

36) 【注意】（非兼容）v-bind 合并行为

a. 概览

a) 不兼容：v-bind 的绑定顺序会影响渲染结果。

37) 【注意】（非兼容）VNode 生命周期事件

a. 概览

a) 在 Vue 2 中，我们可以通过事件来监听组件生命周期中的关键阶段。这些事件名都是以 hook: 前缀开头，并跟随相应的生命周期钩子的名字。
b) 在 Vue 3 中，这个前缀已被更改为 vnode-。额外地，这些事件现在也可用于 HTML 元素，和在组件上的用法一样。

38) 【注意】（非兼容）侦听数组

a. 概览
a) 非兼容: 当侦听一个数组时，只有当数组被替换时才会触发回调。如果你需要在数组被改变时触发回调，必须指定 deep 选项。

4. 其它