

# 1. Vue2高级用法

## 1. 课程资料



0722.1.vue2.zip


167.15KB



## 2. 课程目标



- 初级：
  - 掌握 Vue2 mixin 的使用,Vue3会用 composition API 替代
  - 掌握动画与过渡
  - 掌握插槽的使用
- 中级：
  - 熟悉 mixin 的弊端以及如何处理
  - 了解过渡基本原理
  - 了解插件机制
- 高级：
  - 掌握 Vue 插件化思想
  - 掌握 Vue2 相关高级用法


 目录导航

## 3. 课程大纲

- Mixin 的使用与问题

- Vue 过渡与动画
- 插槽 slot
- 插件化机制

## 4. 课程内容


 注意：我们今天的课程一定要使用 Vue2（v2.7.14），为了帮大家实现 Vue2 到 3 的平滑过渡。

Vue2 的 github 仓库：<https://github.com/vuejs/vue>

```
npm install -g @vue/cli
```

```
vue create 0722.1.vue2
```

### mixin 复用【vue不会用了，了解一下】

 首先说明一点，这也是大家需要具备的一种思考问题的方法——**穿越**

mixin 实现复用的同时带来了很多问题，例如：命名污染、依赖不透明

**Vue3 用 Composition API 替代**

### 基础使用

混入 (mixin) 提供了一种非常灵活的方式，来分发 Vue 组件中的可复用功能。一个混入对象可以包含任意组件选项。当组件使用混入对象时，所有混入对象的选项将被“混合”进入该组件本身的选项。

例子：

```
// 定义一个混入对象
var myMixin = {
  created: function () {
```

```

    this.hello()
  },
  methods: {
    hello: function () {
      console.log('hello from mixin!')
    }
  }
}

// 定义一个使用混入对象的组件
var Component = Vue.extend({
  mixins: [myMixin]
})

var component = new Component() // => "hello from mixin!"

```

## 选项合并

当组件和混入对象含有同名选项时，这些选项将以恰当的方式进行“合并”。

比如，数据对象在内部会进行递归合并，并在发生冲突时以组件数据优先。

```

var mixin = {
  data: function () {
    return {
      message: 'hello',
      foo: 'abc'
    }
  }
}

new Vue({
  mixins: [mixin],
  data: function () {
    return {
      message: 'goodbye',
      bar: 'def'
    }
  },
  created: function () {
    console.log(this.$data)
    // => { message: "goodbye", foo: "abc", bar: "def" }
  }
})

```

同名钩子函数将合并为一个数组，因此都将被调用。另外，混入对象的钩子将在组件自身钩子之前调用。

```
var mixin = {
  created: function () {
    console.log('混入对象的钩子被调用')
  }
}

new Vue({
  mixins: [mixin],
  created: function () {
    console.log('组件钩子被调用')
  }
})

// => "混入对象的钩子被调用"
// => "组件钩子被调用"
```

值为对象的选项，例如 `methods`、`components` 和 `directives`，将被合并为同一个对象。两个对象键名冲突时，取组件对象的键值对。

```
var mixin = {
  methods: {
    foo: function () {
      console.log('foo')
    },
    conflicting: function () {
      console.log('from mixin')
    }
  }
}

var vm = new Vue({
  mixins: [mixin],
  methods: {
    bar: function () {
      console.log('bar')
    },
    conflicting: function () {
      console.log('from self')
    }
  }
})
```

```
vm.foo() // => "foo"  
vm.bar() // => "bar"  
vm.conflicting() // => "from self"
```

注意： `Vue.extend()` 也使用同样的策略进行合并。

## 全局混入

混入也可以进行全局注册。使用时格外小心！一旦使用全局混入，它将影响每一个之后创建的 Vue 实例。使用恰当时，这可以用来为自定义选项注入处理逻辑。

```
// 为自定义的选项 'myOption' 注入一个处理器。  
Vue.mixin({  
  created: function () {  
    var myOption = this.$options.myOption  
    if (myOption) {  
      console.log(myOption)  
    }  
  }  
})  
  
new Vue({  
  myOption: 'hello!'  
})  
// => "hello!"
```

## 细数 mixin 存在的问题

### 命名冲突

我们看到了mixin模式如何在运行时合并两个对象。如果他们两个都共享同名属性，会发生什么？

```
const mixin = {  
  data: () => ({  
    myProp: null  
  })  
}  
  
export default {  
  mixins: [mixin],  
  data: () => ({  
    // 同名！  
  })  
}
```

```
    myProp: null
  })
}
```

这就是合并策略发挥作用的地方。这是一组规则，用于确定当一个组件包含多个具有相同名称的选项时会发生什么。

Vue 组件的默认（但可以配置）合并策略指示本地选项将覆盖 mixin 选项。Vue 组件的默认（可选配置）合并策略指示本地选项将覆盖 mixin 选项。不过也有例外，例如，如果我们有多个相同类型的生命周期钩子，这些钩子将被添加到一个钩子数组中，并且所有的钩子都将被依次调用。

尽管我们不应该遇到任何实际的错误，但是在跨多个组件和 mixin 处理命名属性时，编写代码变得越来越困难。一旦第三方 mixin 作为带有自己命名属性的 npm 包被添加进来，就会特别困难，因为它们可能会导致冲突。


## 依赖不透明

换句话说，依赖不是局部声明式的。

mixin 和使用它的组件之间没有层次关系。这意味着组件可以使用 mixin 中定义的数据属性（例如 myData），但是 mixin 也可以使用假定在组件中定义的数据属性（例如 myData）。以后的某天如果想修改 mixin，包袱有点重。

## vue.js 动画特效 & 常见组件库介绍

### 进入/离开 & 列表过渡

 Vue 在插入、更新或者移除 DOM 时，提供多种不同方式的应用过渡效果。包括以下工具：

- 在 CSS 过渡和动画中自动应用 class
- 可以配合使用第三方 CSS 动画库，如 Animate.css
- 在过渡钩子函数中使用 JavaScript 直接操作 DOM
- 可以配合使用第三方 JavaScript 动画库，如 Velocity.js、gsap

Vue 提供了 `transition` 的封装组件，在下列情形中，可以给任何元素和组件添加进入/离开过渡

- 条件渲染 (使用 `v-if`)
- 条件展示 (使用 `v-show`)
- 动态组件

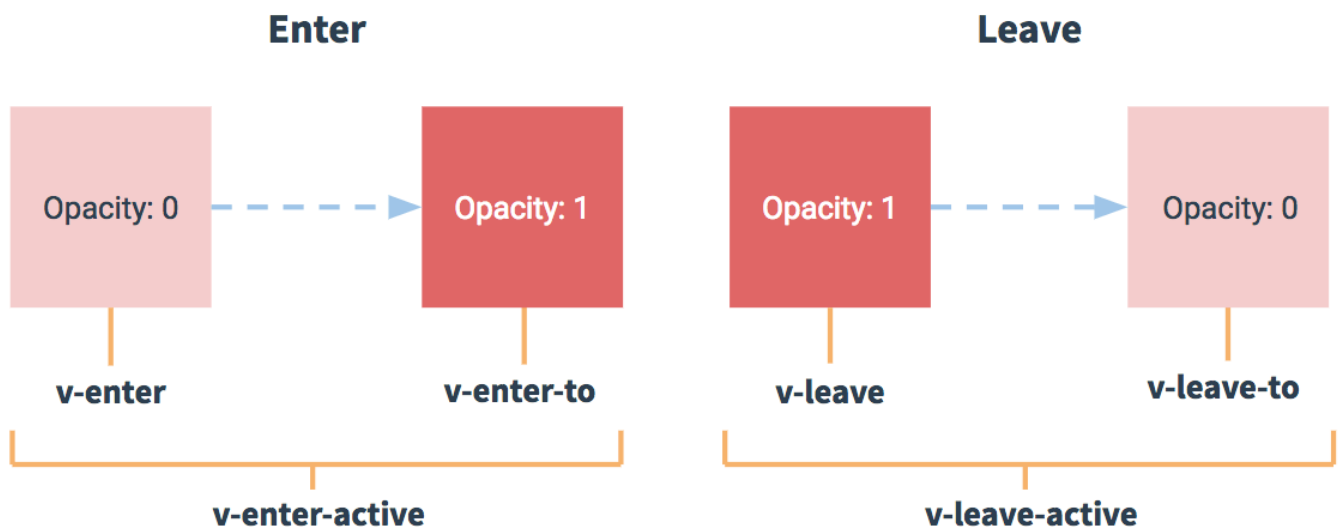
- 组件根节点

## 基础使用示例

```
<div id="demo">
  <button v-on:click="show = !show">
    Toggle
  </button>
  <transition name="fade">
    <p v-if="show">hello</p>
  </transition>
</div>
```

```
.fade-enter-active, .fade-leave-active {
  transition: opacity .5s;
}
.fade-enter, .fade-leave-to /* .fade-leave-active below version 2.1.8 */ {
  opacity: 0;
}
```

```
export default {
  data() {
    show: true
  }
}
```



## 自定义过渡类名

```
<link href="https://cdn.jsdelivr.net/npm/animate.css@3.5.1" rel="stylesheet" type="text/css">
```

```
<div id="example-3">
  <button @click="show = !show">
    Toggle render
  </button>
  <transition
    name="custom-classes-transition"
    enter-active-class="animated tada"
    leave-active-class="animated bounceOutRight"
  >
    <p v-if="show">hello</p>
  </transition>
</div>
```

## 动画钩子

```
<transition
  v-on:before-enter="beforeEnter"
  v-on:enter="enter"
  v-on:after-enter="afterEnter"
  v-on:enter-cancelled="enterCancelled">
```



```

    v-on:before-leave="beforeLeave"
    v-on:leave="leave"
    v-on:after-leave="afterLeave"
    v-on:leave-cancelled="leaveCancelled"
  >
    <!-- ... -->
</transition>

```

## 多组件过渡与列表过渡

```

<div id="list-demo" class="demo">
  <button v-on:click="add">Add</button>
  <button v-on:click="remove">Remove</button>
  <transition-group name="list" tag="p">
    <span v-for="item in items" v-bind:key="item" class="list-item">
      {{ item }}
    </span>
  </transition-group>
</div>

```

```

{
  data: {
    items: [1,2,3,4,5,6,7,8,9],
    nextNum: 10
  },
  methods: {
    randomIndex: function () {
      return Math.floor(Math.random() * this.items.length)
    },
    add: function () {
      this.items.splice(this.randomIndex(), 0, this.nextNum++)
    },
    remove: function () {
      this.items.splice(this.randomIndex(), 1)
    },
  }
}

```

```

.list-item {
  display: inline-block;

```

```

    margin-right: 10px;
  }
  .list-enter-active, .list-leave-active {
    transition: all 1s;
  }
  .list-enter, .list-leave-to
/* .list-leave-active for below version 2.1.8 */ {
    opacity: 0;
    transform: translateY(30px);
  }

```

## 状态过渡

这个概念其实很好理解，就是通过状态去驱动视图更新从而实现动画过渡。

在这里我们首先要了解**计算属性与数据监听（computed、watch）**。

```

<script src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.2.4/gsap.min.js">
</script>

<div id="animated-number-demo">
  <input v-model.number="number" type="number" step="20">
  <p>{{ animatedNumber }}</p>
</div>

```

```

{
  data: {
    number: 0,
    tweenedNumber: 0
  },
  computed: {
    animatedNumber: function() {
      return this.tweenedNumber.toFixed(0);
    }
  },
  watch: {
    number: function(newValue) {
      gsap.to(this.$data, { duration: 0.5, tweenedNumber: newValue });
    }
  }
}

```

```
→ ↻ github.com/vuejs/vue/blob/49b6bd4264c25ea41408f066a1835f38bf6fe9f1/src/platforms/web/runtime/components/transition-group.t
49b6bd4 vue / src / platforms / web / runtime / components / transition-group.ts
Code Blame 204 lines (185 loc) · 6.2 KB
104     updated() {
120       this._reflow = document.body.offsetHeight
121
122       children.forEach((c: VNode) => {
123         if (c.data!.moved) {
124           const el: any = c.elm
125           const s: any = el.style
126           addTransitionClass(el, moveClass)
127           s.transform = s.WebkitTransform = s.transitionDuration = ''
128           el.addEventListener(
129             transitionEndEvent,
130             (el._moveCb = function cb(e) {
131               if (e && e.target !== el) {
132                 return
133               }
134               if (!e || /transform$/.test(e.propertyName)) {
135                 el.removeEventListener(transitionEndEvent, cb)
136                 el._moveCb = null
137                 removeTransitionClass(el, moveClass)
138               }
139             })
140         )
141       }
142     })
}
```

## 常用动画相关库

大家可以在工作中选用以下动画相关库实现动效。

**gsap**

animated.css

tween.js

等

作业：小实战

案例：创建一个简单的待办事项列表，点击添加按钮时，新的待办事项会以淡入的动画效果出现；当点击删除按钮时，待办事项会以淡出的动画效果消失。

1. 创建一个基本的待办事项列表组件 `TodoList`，包含以下内容：

- 一个输入框和一个添加按钮，用于输入待办事项并添加到列表中；
- 一个待办事项列表，用于展示已添加的待办事项；
- 每个待办事项具有删除按钮，点击时可以删除该事项。

2. 使用Vue2的transition组件实现动画效果：

- 在待办事项列表组件的模板中，使用 `<transition>` 包裹待办事项列表；
- 在待办事项列表的每个待办事项项中，使用 `<transition-group>` 包裹，并设置 `name` 属性为 'fade'；
- 在 CSS 样式中定义 'fade' 动画效果，包括进入和离开的过渡效果。

3. 完成添加和删除待办事项的功能：

- 在 `TodoList` 组件的 data 中定义一个 `todoList` 数组，用于存储待办事项；
- 在添加按钮的点击事件中，将输入框的值添加到 `todoList` 数组中，并清空输入框；
- 在每个待办事项项的删除按钮的点击事件中，根据索引从 `todoList` 数组中移除对应的待办事项。

4. 在 CSS 样式中定义 'fade' 动画效果：

- 使用 Vue2 默认的过渡类名： `v-enter`、 `v-enter-active`、 `v-leave`、 `v-leave-active`；
- 定义进入的过渡效果为淡入： `opacity: 0;;`
- 定义离开的过渡效果为淡出： `opacity: 0;;`

这样，当你输入待办事项并点击添加按钮时，新的待办事项会以淡入的动画效果出现；当点击删除按钮时，待办事项会以淡出的动画效果消失。

8  
16  
4  
5  
14

```
Elements Console Sources Network Performance Memory Ap
<div data-v-2e1e8f08>12</div>
<div data-v-2e1e8f08>10</div>
<div data-v-2e1e8f08>8</div>
<div data-v-2e1e8f08 class="list-enter-active list-enter-to">
16</div>
<div data-v-2e1e8f08>4</div>
<div data-v-2e1e8f08>5</div>
<div data-v-2e1e8f08>14</div> == $0
</span>
```

## 应用场景

1. 组件中的动效
2. 路由动画

如果实在 react 中，我们可以选用 react-spring，思路类似

## 插槽

### 插槽基本使用

插槽的概念可以与 react 中 renderProps 对比。

假设我们现在有这样的需求

```
<div class="container">
  <header>
    <!-- 我们希望把页头放这里 -->
  </header>
  <main>
    <!-- 我们希望把主要内容放这里 -->
  </main>
  <footer>
    <!-- 我们希望把页脚放这里 -->
  </footer>
</div>
```

我们在编写代码时，组件内部并不清楚这块内容的具体实现，我就需要将这个坑位留出，需要开发者传进来。

我们为每个插槽取个名字

```
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
```

## 插槽中使用数据

有时让插槽内容能够访问子组件中才有的数据是很有用的。例如，设想一个带有如下模板的

`<current-user>` 组件：

```
<span>
  <slot>{{ user.lastName }}</slot>
</span>
```

我们可能想换掉备用内容，用名而非姓来显示。如下：

```
<current-user>
  {{ user.firstName }}
</current-user>
```

然而上述代码不会正常工作，因为只有 `<current-user>` 组件可以访问到 `user`，而我们提供的内容是在父级渲染的。

为了让 `user` 在父级的插槽内容中可用，我们可以将 `user` 作为 `<slot>` 元素的一个 attribute 绑定上去：

```
<span>
  <slot v-bind:user="user">
    {{ user.lastName }}
  </slot>
</span>
```

多个插槽与数据，我们可以这样实现

```
<template>
  <div class="slot-demo">
    <slot name="demo">this is demo slot.</slot>
    <slot text="this is a slot demo , " :msg="msg"></slot>
  </div>
</template>

<script>
export default {
  name: 'SlotDemo',
  data () {
    return {
      msg: 'this is scoped slot content.'
    }
  }
}
</script>
```

```
<template>
  <slot-demo>
    <template v-slot:demo>this is custom slot.</template>
    <template v-slot="scope">
      <p>{{ scope.text }}{{ scope.msg }}</p>
    </template>
  </slot-demo>
</template>
```

插槽的原理【这个好好了解一下】

<https://github.com/vuejs/vue>

vm.\$slots

挑几个重点方法：

- \$slots & \$scopedSlots
- renderSlot
- processSlot、processSlotContent
- generate

插槽的本质就是函数！

<https://github.com/vuejs/vue/blob/49b6bd4264c25ea41408f066a1835f38bf6fe9f1/src/core/instance/render-helpers/render-slot.ts#L7>

## 插件

插件可以是对象，或者是一个函数。如果是对象，那么对象中需要提供 **install** 函数，如果是函数，形态需要跟前面提到的 **install** 函数保持一致。

install 是组件安装的一个方法，跟 npm install 完全不一样，npm install 是一个命令

## 定义插件

```
const MyPlugin = {
  install(Vue, options) {
    // 1. 添加全局方法或 property
    Vue.myGlobalMethod = function () {
      // 逻辑...
    }

    // 2. 添加全局资源
    Vue.directive('my-directive', {
      bind (el, binding, vnode, oldVnode) {
        // 逻辑...
      }
      ...
    })

    // 3. 注入组件选项
    Vue.mixin({
      created: function () {
        // 逻辑...
      }
      ...
    })
  }
}
```



```

    // 4. 添加实例方法
    Vue.prototype.$myMethod = function (methodOptions) {
      // 逻辑...
    }
  }
};

```

## 使用插件

```

Vue.use(MyPlugin);

{{ $myMethod }}

```

## 插件化机制原理

```

export function initUse (Vue: GlobalAPI) {
  Vue.use = function (plugin: Function | Object) {
    // 获取已经安装的插件
    const installedPlugins = (this._installedPlugins || (this._installedPlugins
= []))
    // 看看插件是否已经安装，如果安装了直接返回
    if (installedPlugins.indexOf(plugin) > -1) {
      return this
    }

    // toArray(arguments, 1)实现的功能就是，获取Vue.use(plugin,xx,xx)中的其他参数。
    // 比如 Vue.use(plugin,{size:'mini', theme:'black'})，就会回去到plugin意外的参
数
    const args = toArray(arguments, 1)
    // 在参数中第一位插入Vue，从而保证第一个参数是Vue实例
    args.unshift(this)
    // 插件要么是一个函数，要么是一个对象(对象包含install方法)
    if (typeof plugin.install === 'function') {
      // 调用插件的install方法，并传入Vue实例
      plugin.install.apply(plugin, args)
    } else if (typeof plugin === 'function') {
      plugin.apply(null, args)
    }
    // 在已经安装的插件数组中，放进去
    installedPlugins.push(plugin)
    return this
  }
}

```

```
}  
}
```

## 具体实践

### Vue-Router

```
import View from './components/view'  
import Link from './components/link'  
  
export let _Vue  
  
export function install (Vue) {  
  if (install.installed && _Vue === Vue) return  
  install.installed = true  
  
  _Vue = Vue  
  
  const isDef = v => v !== undefined  
  
  const registerInstance = (vm, callVal) => {  
    let i = vm.$options._parentVnode  
    if (isDef(i) && isDef(i = i.data) && isDef(i = i.registerRouteInstance)) {  
      i(vm, callVal)  
    }  
  }  
  
  Vue.mixin({  
    beforeCreate () {  
      if (isDef(this.$options.router)) {  
        this._routerRoot = this  
        this._router = this.$options.router  
        this._router.init(this)  
        Vue.util.defineReactive(this, '_route', this._router.history.current)  
      } else {  
        this._routerRoot = (this.$parent && this.$parent._routerRoot) || this  
      }  
      registerInstance(this, this)  
    },  
    destroyed () {  
      registerInstance(this)  
    }  
  })  
  
  Object.defineProperty(Vue.prototype, '$router', {
```

```
    get () { return this._routerRoot._router }
  })

Object.defineProperty(Vue.prototype, '$route', {
  get () { return this._routerRoot._route }
})

Vue.component('RouterView', View)
Vue.component('RouterLink', Link)

const strats = Vue.config.optionMergeStrategies
// use the same hook merging strategy for route hooks
strats.beforeRouteEnter = strats.beforeRouteLeave = strats.beforeRouteUpdate
= strats.created
}
```

