# Vue3小兔鲜电商前台



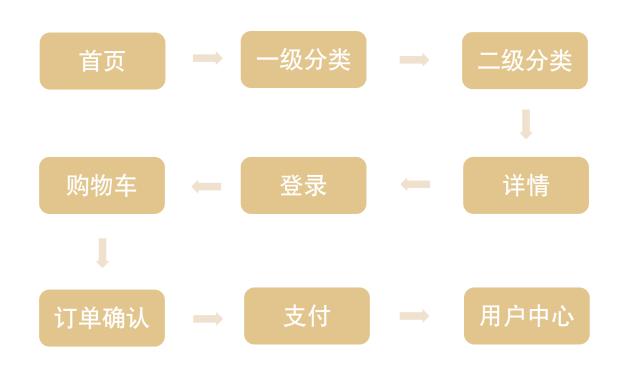


## 业务模块



新鲜好物 新鲜出炉品质靠谱







## 核心技术栈

Vue2全家桶

vue-cli

Vuex

ElementUI

Vue2 + Options

Vue-Router

Vue3全家桶

create-vue

Pinia

ElementPlus

Vue3 + Setup

Vue-Router



VueUse



## 业务(亮点)解决方案

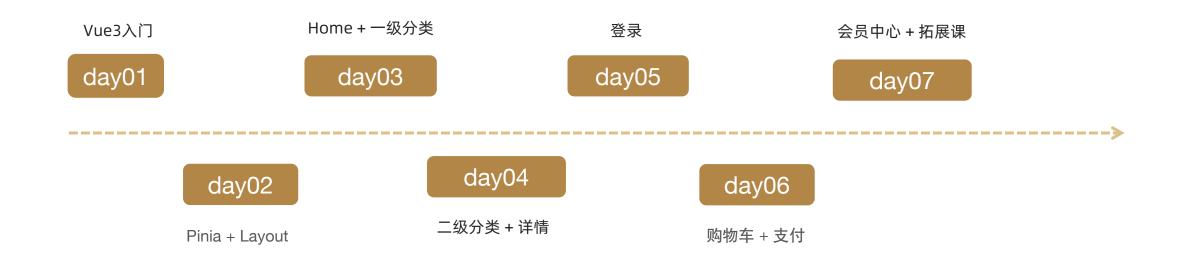
- 1. 长页面吸顶交互实现
- 2. 图片懒加载指令封装
- 3. 面板插槽组件等业务通用组件封装
- 4. SKU电商组件封装
- 5. 通用逻辑函数封装
- 6. 路由缓存问题处理
- 7. 基于业务逻辑的组件拆分思想
- 8. 支付宝三方支付

...





## 课程安排

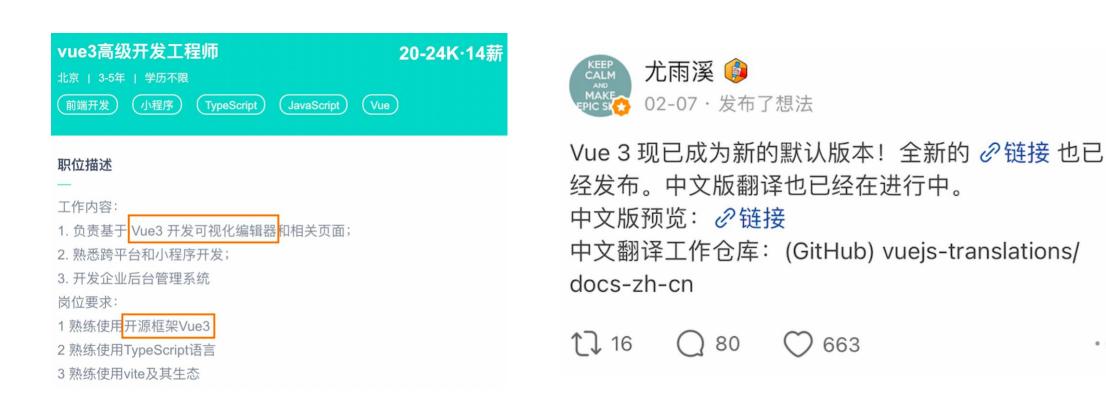








### 为什么需要学Vue3?



0 (



## Vue3组合式API体验

通过一个 Counter案例 体验Vue3新引入的组合式API



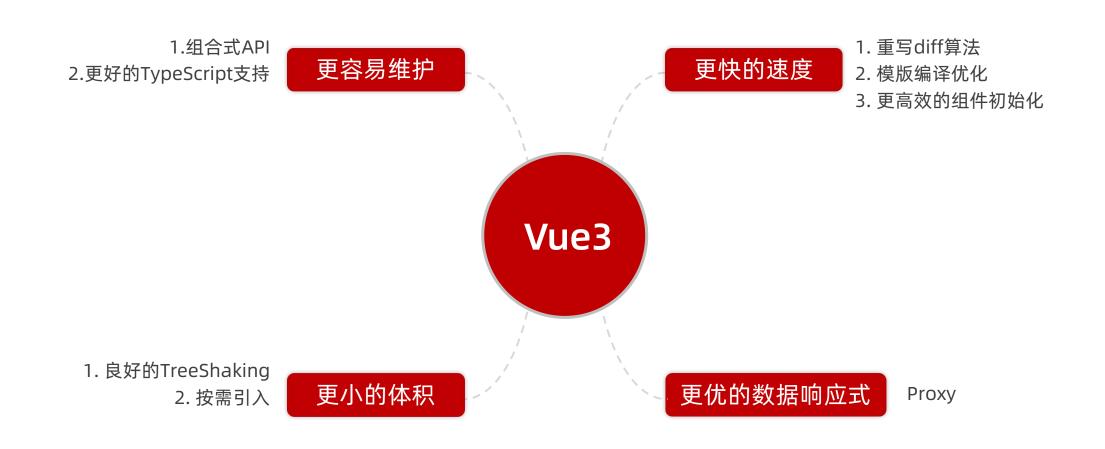
```
1 <script>
2 export default {
3   data () {
4    return {
5       count: 0
6    }
7    },
8   methods: {
9    addCount () {
10       this.count++
11    }
12    }
13 }
14 </script>
```

```
1 <script setup>
2 import { ref } from 'vue'
3 const count = ref(0)
4 const addCount = () => count.value++
5 </script>
```

1. 代码量变少了 2. 分散式维护转为集中式维护



## Vue3更多的优势





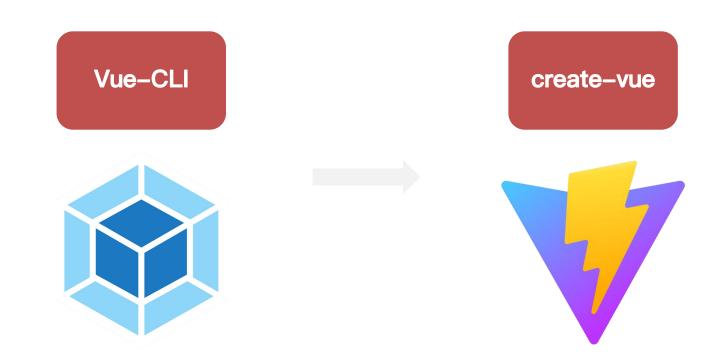


## 使用create-vue搭建Vue3项目



### 认识 create-vue

create-vue是Vue官方新的脚手架工具,底层切换到了 vite(下一代前端工具链),为开发提供极速响应





## 使用create-vue创建项目

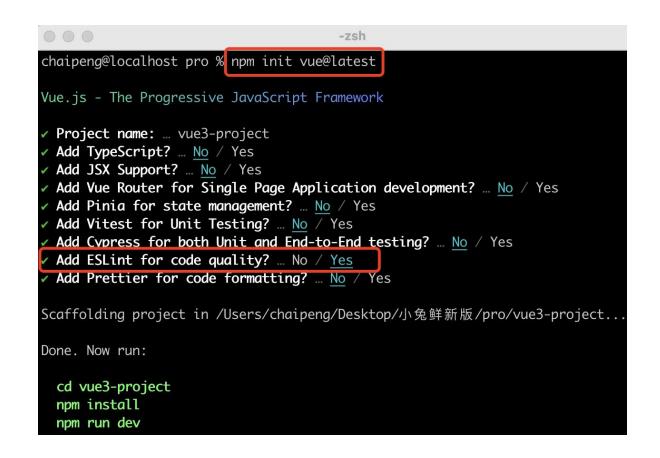
1. 前提环境条件

已安装 16.0 或更高版本的 Node.js

2. 创建一个Vue应用

npm init vue@latest

这一指令将会安装并执行 create-vue



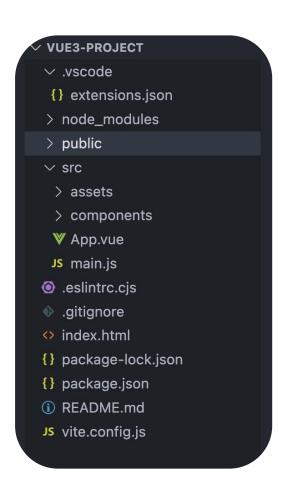




## 熟悉项目目录和关键文件



### 项目目录和关键文件



#### 关键文件:

- 1. vite.config.js 项目的配置文件 基于vite的配置
- 2. package.json 项目包文件 核心依赖项变成了 Vue3.x 和 vite
- 3. main.js 入口文件 createApp函数创建应用实例
- 4. app.vue 根组件 SFC单文件组件 script template style

变化一: 脚本script和模板template顺序调整

变化二:模板template不再要求唯一根元素

变化三: 脚本script添加setup标识支持组合式API

5. index.html - 单页入口 提供id为app的挂载点



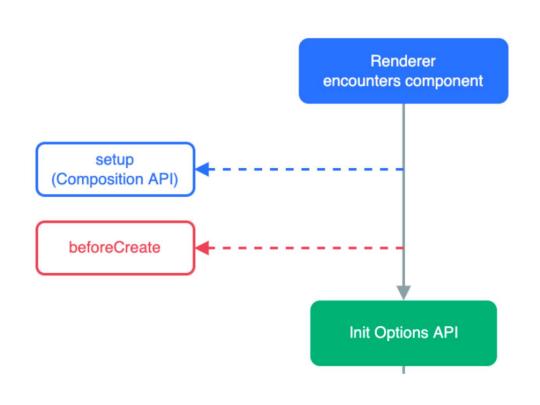


## 04 组合式API – setup选项



## setup选项的写法和执行时机

```
<script>
export default {
 setup () {
  beforeCreate () {
</script>
```





## setup选项中写代码的特点

```
<script>
   export default {
     setup () {
      // 数据
      const message = 'this is message'
      // 函数
       const logMessage = () => {
         console.log(message)
       return {
         message,
         logMessage
16 </script>
```

```
1 <template>
2 <!-- 使用数据和方法 -->
3 {{ message }}
4 <button @click="logMessage">
5 log message
6 </button>
7 </template>
```



## <script setup> 语法糖

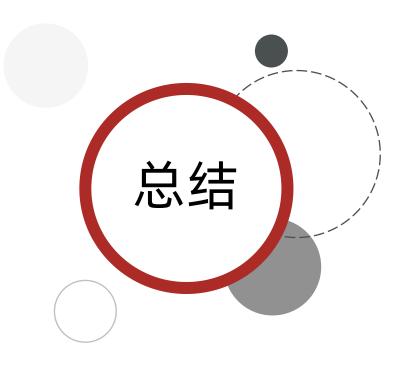
#### 原始复杂写法

```
<script>
   export default {
     setup () {
      // 数据
      const message = 'this is message'
      // 函数
      const logMessage = () => {
         console.log(message)
       return {
         message,
         logMessage
15 }
16 </script>
```

#### 语法糖写法

```
1 <script setup>
2 // 数据
3 const message = 'this is message'
4 // 函数
5 const logMessage = () => {
6 console.log(message)
7 }
8 </script>
```





- 1. setup选项的执行时机? beforeCreate钩子之前自动执行
- 2. setup写代码的特点是什么? 定义数据 + 函数 然后以对象方式return
- 3. <script setup>解决了什么问题? 经过语法糖的封装更简单的使用组合式API
- 4. setup中的this还指向组件实例吗? 指向undefined





05 组合式API – reactive和ref函数



## reactive()

作用: 接受对象类型数据的参数传入并返回一个响应式的对象

#### 核心步骤:

```
1 <script setup>
2 // 导入
3 import { reactive } from 'vue'
4
5 // 执行函数 传入参数 变量接收
6 const state = reactive(对象类型数据)
7
8 </script>
```

- 1. 从 vue 包中导入 reactive 函数
- 2. 在 <script setup> 中执行 reactive 函数并传入类型为对象的初始值,并使用变量接收返回值



## ref()

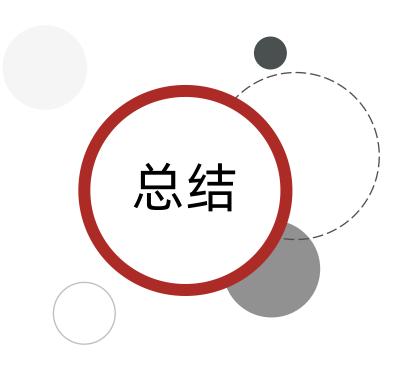
作用: 接收简单类型或者对象类型的数据传入并返回一个响应式的对象

#### 核心步骤:

```
<script setup>
// 导入
import { ref } from 'vue'
// 执行函数 传入参数 变量接收
const count = ref(简单类型或者复杂类型数据)
</script>
```

- 1. 从 vue 包中导入 ref 函数
- 2. 在 <script setup> 中执行 ref 函数并传入初始值,使用变量接收 ref 函数的返回值





- 1. reactive和ref函数的共同作用是什么? 用函数调用的方式生成响应式数据
- 2. reactive vs ref?
  - 1. reactive不能处理简单类型的数据
  - 2. ref参数类型支持更好但是必须通过.value访问修改
  - 3. ref函数的内部实现依赖于reactive函数
- 3. 在实际工作中推荐使用哪个? 推荐使用ref函数,更加灵活,小兔鲜项目主用ref





## 06 组合式API – computed



## computed计算属性函数

计算属性基本思想和Vue2的完全一致,组合式API下的计算属性只是修改了写法

核心步骤: 1. 导入computed函数

2. 执行函数 在回调参数中return基于响应式数据做计算的值,用变量接收

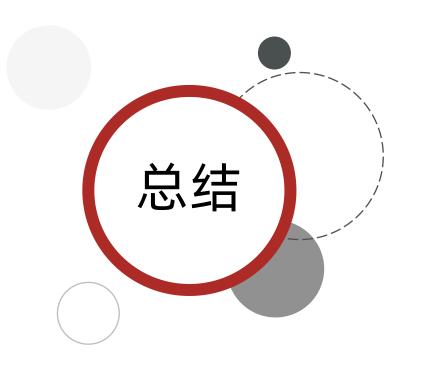
```
1 <script setup>
2 // 导入
3 import { computed } from 'vue'
4 // 执行函数 变量接收 在回调参数中return计算值
5 const computedState = computed(() => {
6 return 基于响应式数据做计算之后的值
7 })
8 </script>
```



## 计算属性小案例

计算公式: 始终从原始响应式数组中筛选出大于2的所有项 - filter





## 最佳实践

1. 计算属性中不应该有"副作用"

比如异步请求/修改dom

2. 避免直接修改计算属性的值

计算属性应该是只读的





07 组合式API – watch



## watch函数

vue	
vue	
vuex	
vue-router	

作用: 侦听一个或者多个数据的变化,数据变化时执行回调函数

俩个额外参数: 1. immediate (立即执行) 2. deep (深度侦听)



## 基础使用 - 侦听单个数据

- 1. 导入watch函数
- 2. 执行watch函数传入要侦听的响应式数据(ref对象)和回调函数

```
<script setup>
   // 1. 导入watch
   import { ref, watch } from 'vue'
   const count = ref(0)
   // 2. 调用watch 侦听变化
   watch(count, (newValue, oldValue) => {
     console.log(`count发生了变化,老值为${oldValue},新值为${newValue}`)
10 })
12 </script>
```



## 基础使用 - 侦听多个数据

说明:同时侦听多个响应式数据的变化,不管哪个数据变化都需要执行回调

```
<script setup>
   import { ref, watch } from 'vue'
   const count = ref(0)
   const name = ref('cp')
   // 侦听多个数据源
8 watch(
     [count, name],
     ([newCount, newName], [oldCount, oldName]) => {
       console.log('count或者name变化了', [newCount, newName], [oldCount, oldName])
13 )
15 </script>
```



#### immediate

说明:在侦听器创建时立即触发回调,响应式数据变化之后继续执行回调

Please input vue element cooking mint-ui vuex vue-router babel

```
const count = ref(0)
  watch(count, () => {
    console.log('count发生了变化')
  }, {
    immediate: true
6 })
```



## deep

默认机制:通过watch监听的ref对象默认是<mark>浅层侦听的,直接修改嵌套的对象属性不会触发回调执行,</mark>需要开启deep 选项

```
const state = ref({ count: 0 })
watch(state, () => console.log('数据变化了'))
const changeStateByCount = () => {
 // 直接修改属性 -> 不会触发回调
 state.value.count++
```



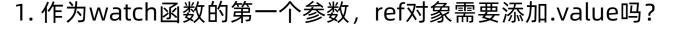
## 精确侦听对象的某个属性

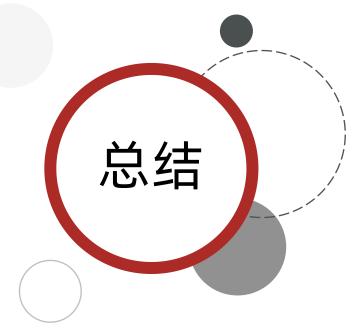
需求:在不开启deep的前提下,侦听age的变化,只有age变化时才执行回调

```
const info = ref({
  name: 'cp',
  age: 18
4 })
5
```

```
1 const info = ref({
2    name: 'cp',
3    age: 18
4 })
5
6 watch(
7    () => info.value.age,
8    () => console.log('age发生变化了')
9 )
```







#### 不需要, watch会自动读取

2. watch只能侦听单个数据吗?

#### 单个或者多个

3. 不开启deep,直接修改嵌套属性能触发回调吗?

不能,默认是浅层侦听

4. 不开启deep, 想在某个层次比较深的属性变化时执行回调怎么做?

可以把第一个参数写成函数的写法,返回要监听的具体属性





组合式API – 生命周期函数



## Vue3的生命周期API(选项式 VS 组合式)

选项式API	组合式API
beforeCreate/created	setup
beforeMount	onBeforeMount
mounted	onMounted
beforeUpdate	onBeforeUpdate
updated	onUpdated
beforeUnmount	onBeforeUnmount
unmounted	onUnmounted



## 生命周期函数基本使用

- 1. 导入生命周期函数
- 2. 执行生命周期函数 传入回调

```
    import { onMounted } from 'vue'
    onMounted(() => {
    // 自定义逻辑
    })
```



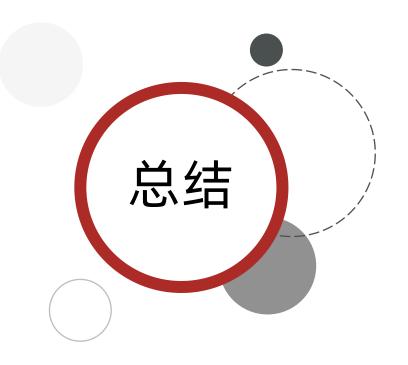
## 执行多次

生命周期函数是可以执行多次的,多次执行时传入的回调会在时机成熟时依次执行

```
onMounted(() => {
console.log('mount1')
}

onMounted(() => {
onMounted(() => {
console.log('mount2')
})
```





- 1. 组合式API中生命周期函数的格式是什么? on + 生命周期名字
- 组合式API中可以使用onCreated吗?
   没有这个钩子函数,直接写到setup中
- 3. 组合式API中组件卸载完毕时执行哪个函数? onUnmounted





# 09 组合式API – 父子通信



## 组合式API下的父传子

#### 基本思想

- 1. 父组件中给子组件绑定属性
- 2. 子组件内部通过props选项接收

```
□ …
                                                       ▼ son-com.vue ×
                    父组件
                                                                                 子组件
src > ♥ App.vue > ...
                                                        src > ♥ son-com.vue > ...
                                                              <script setup>
      <script setup>
      // 引入子组件
                                                             // 2. 通过 defineProps "编译器宏" 接收子组件传递的数据
      import sonComVue from './son-com.vue'
                                                              const props = defineProps({
      </script>
                                                                message: String
                                                              })
      <template>
                                                              </script>
        <!-- 1. 绑定属性 message -->
        <sonComVue message="this is app message"</pre>
                                                              <template>
      </template>
                                                                {{ message }}
                                                              </template>
```



## 组合式API下的子传父

#### 基本思想

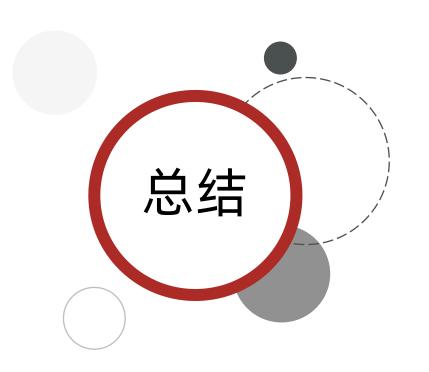
- 1. 父组件中给子组件标签通过@绑定事件
- 2. 子组件内部通过 \$emit 方法触发事件

```
▼ App.vue ×

▼ son-com.vue ×

                      父组件
                                                                                 子组件
src > ♥ App.vue > ...
                                                        src > ♥ son-com.vue > ...
      <script setup>
                                                              <script setup>
                                                              // 2. 通过 defineEmits编译器宏生成emit方法
      // 引入子组件
      import sonComVue from './son-com.vue'
                                                              const emit = defineEmits(['get-message'])
      const getMessage = (msg) => {
        console.log(msg)
                                                              const sendMsg = () => {
                                                                // 3. 触发自定义事件 并传递参数
      </script>
                                                                emit('get-message', 'this is son msg')
      <template>
                                                              </script>
     <!-- 1. 绑定自定义事件 -->
        <sonComVue @get-message="getMessage" />
                                                              <template>
      </template>
                                                                <button @click="sendMsg">sendMsg</button>
                                                              </template>
```





## 父传子

- 1. 父传子的过程中通过什么方式接收props? defineProps({属性名:类型})
- setup语法糖中如何使用父组件传过来的数据?
   const props = defineProps({属性名: 类型})

## 子传父

1. 子传父的过程中通过什么方式得到emit方法? defineEmits(['事件名称'])





# 组合式API – 模版引用



## 模板引用的概念

通过ref标识获取真实的dom对象或者组件实例对象







### 如何使用(以获取dom为例组件同理)

```
<script setup>
   import { ref } from 'vue'
3 // 1. 调用ref函数得到ref对象
4 const h1Ref = ref(null)
   </script>
   <template>
    <!-- 2. 通过ref标识绑定ref对象 -->
    <h1 ref="h1Ref">我是dom标签h1</h1>
   </template>
```

- 1. 调用ref函数生成一个ref对象
- 2. 通过ref标识绑定ref对象到标签



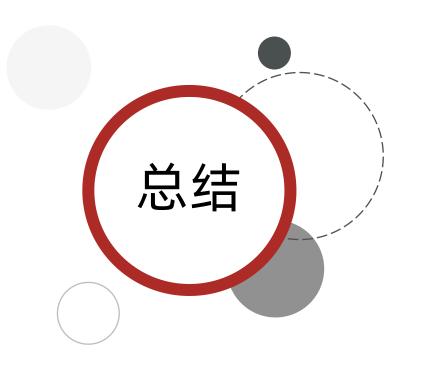
## defineExpose()

默认情况下在<script setup>语法糖下组件内部的属性和方法是不开放给父组件访问的,可以通过defineExpose编译宏指定哪些属性和方法允许访问

```
1 <script setup>
2 import { ref } from 'vue'
3 const testMessage = ref('this is test msg')
4 </script>
5
```

```
1 <script setup>
2 import { ref } from 'vue'
3 const testMessage = ref('this is test msg')
4 defineExpose({
5   testMessage
6 })
7 </script>
```





1. 获取模板引用的时机是什么?

组件挂载完毕

2. defineExpose编译宏的作用是什么?

显式暴露组件内部的属性和方法





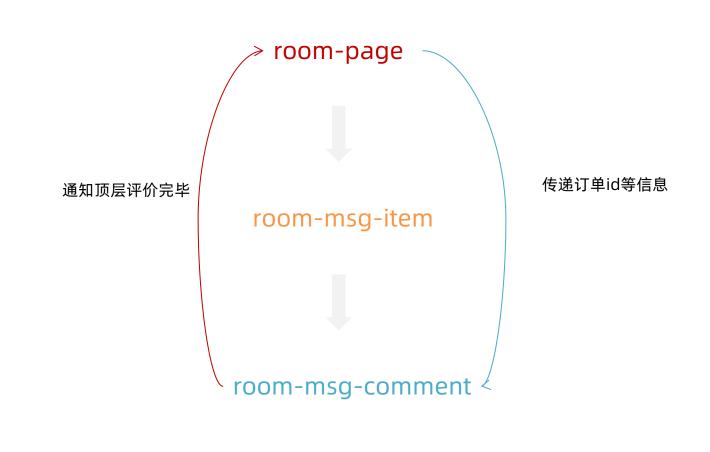
10 组合式API – provide和inject



## 作用和场景

顶层组件向任意的底层组件传递数据和方法,实现跨层组件通信

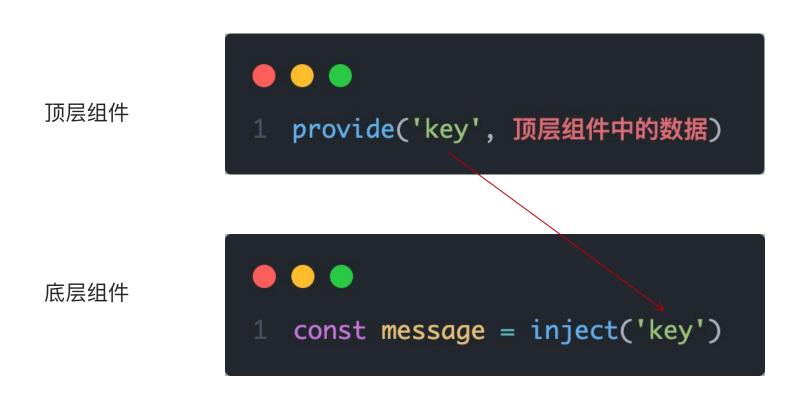






## 跨层传递普通数据

- 1. 顶层组件通过provide函数提供数据
- 2. 底层组件通过inject函数获取数据





## 跨层传递响应式数据

在调用provide函数时,第二个参数设置为ref对象





## 跨层传递方法

顶层组件可以向底层组件传递方法, 底层组件调用方法修改顶层组件中的数据

```
const setCount = () => {
count.value++
}

provide('setCount-key', setCount)
```

```
const setCount = inject('setCount-key')
```

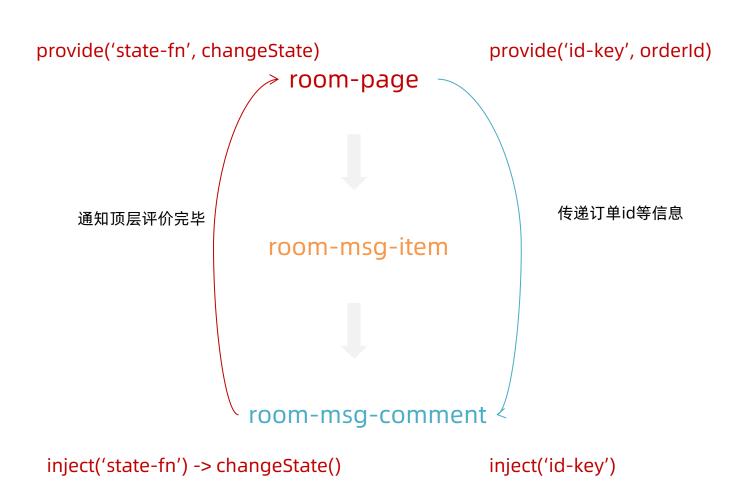
顶层组件

底层组件

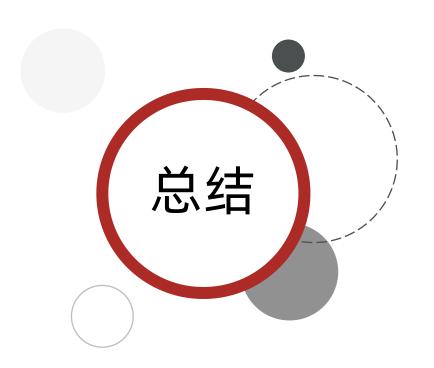


## 需求解决思考









1. provide和inject的作用是什么?

#### 跨层组件通信

2. 如何在传递的过程中保持数据响应式?

第二个参数传递ref对象

3. 底层组件想要通知顶层组件做修改,如何做?

传递方法,底层组件调用方法

4. 一颗组件树中只有一个顶层或底层组件吗?

相对概念,存在多个顶层和底层的关系







# 案例演示

ID	姓名	籍贯	操作	
610000197609132261	阎芳	西藏自治区	编辑	删除
53000020180413347X	蒋娟	海外 海外 –	编辑	删除
820000201102032788	姚丽	甘肃省 天水市 秦安县	编辑	删除
150000198409112737	侯秀兰	澳门特别行政区 澳门半岛 –	编辑	删除
500000199205220920	常勇	西藏自治区 山南地区 曲松县	编辑	删除
710000199005072141	常磊	海外 海外 –	编辑	删除
320000200801152540	秦洋	西藏自治区 拉萨市 林周县	编辑	删除
130000201801054181	蔡娟	上海 上海市 浦东新区	编辑	删除
120000198412037955	周明	广东省 中山市 –	编辑	删除
330000201009206548	于军	内蒙古自治区 兴安盟 扎赉特旗	编辑	删除



## 案例前置说明

### 项目地址

git clone http://git.itcast.cn/heimaqianduan/vue3-basic-project.git

- 1. 模版已经配置好了案例必须的安装包
- 2. 案例用到的接口在 README.MD文件 中
- 3. 案例项目有俩个分支,main主分支为开发分支,complete分支为完成版分支供开发完参考



传智教育旗下高端IT教育品牌