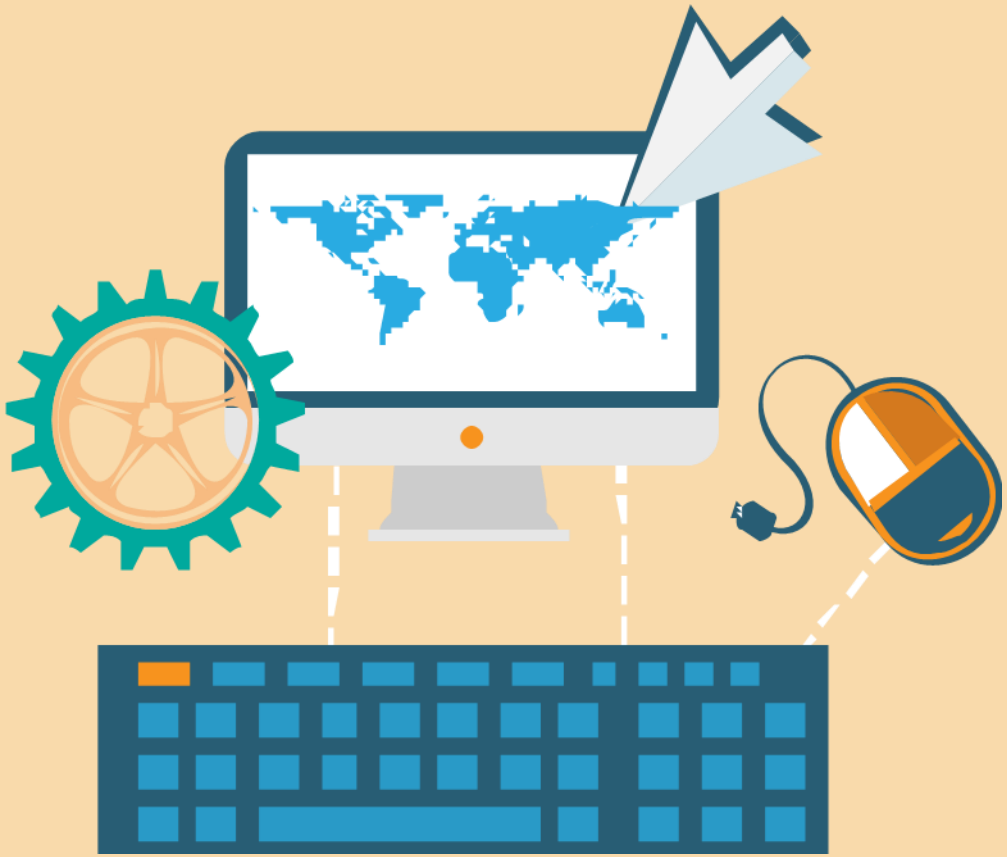
Vue3 – Composition API

王红元 coderwhy

目录
content



1 computed函数使用

2 组件的生命周期函数

3 Provide/Inject使用

4 watch/watchEffect

5 自定义Hook练习

6 script setup语法糖

computed

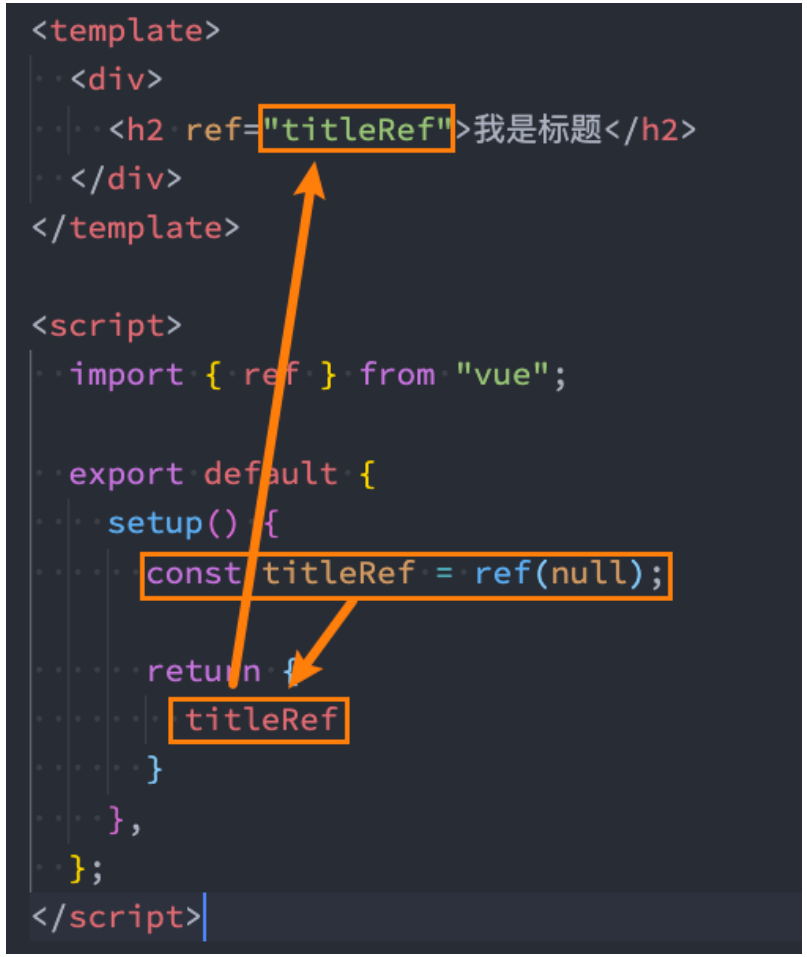
- **在前面我们讲解过计算属性computed：当我们的某些属性是依赖其他状态时，我们可以使用计算属性来处理**
  - 在前面的Options API中，我们是使用computed选项来完成的；
  - 在Composition API中，我们可以在 setup 函数中使用 computed 方法来编写一个计算属性；
- **如何使用computed呢?**
  - 方式一：接收一个getter函数，并为 getter 函数返回的值，返回一个不变的 ref 对象；
  - 方式二：接收一个具有 get 和 set 的对象，返回一个可变的（可读写）ref 对象；

```
const fullName = computed(() => {
  return firstName.value + " " + lastName.value;
})
```

```
const fullName = computed({
  get: () => {
    return firstName.value + " " + lastName.value;
  },
  set: newValue => {
    const names = newValue.split(" ");
    firstName.value = names[0];
    lastName.value = names[1];
  }
})
```

setup中使用ref

- **在setup中如何使用ref获取元素或者组件?**
  - 其实非常简单，我们只需要定义一个ref对象，绑定到元素或者组件的ref属性上即可；



生命周期钩子

- 我们前面说过 setup 可以用来替代 data 、 methods 、 computed 等等这些选项，也可以替代 **生命周期钩子**。
- 那么setup中如何使用生命周期函数呢?
  - 可以使用直接导入的 onX 函数注册生命周期钩子;

```
onMounted(() => {
  console.log("onMounted")
})

onUpdated(() => {
  console.log('onUpdate')
})

onUnmounted(() => {
  console.log('onUnmounted')
})
```

选项式 API	Hook inside `setup`
`beforeCreate`	Not needed*
`created`	Not needed*
`beforeMount`	`onBeforeMount`
`mounted`	`onMounted`
`beforeUpdate`	`onBeforeUpdate`
`updated`	`onUpdated`
`beforeUnmount`	`onBeforeUnmount`
`unmounted`	`onUnmounted`
`activated`	`onActivated`
`deactivated`	`onDeactivated`

TIP

因为 `setup` 是围绕 `beforeCreate` 和 `created` 生命周期钩子运行的，所以不需要显式地定义它们。换句话说，在这些钩子中编写的任何代码都应该直接在 `setup` 函数中编写。

Provide函数

- **事实上我们之前还学习过Provide和Inject，Composition API也可以替代之前的 Provide 和 Inject 的选项。**
- **我们可以通过 provide来提供数据：**
  - 可以通过 provide 方法来定义每个 Property；
- **provide可以传入两个参数：**
  - name：提供的属性名称；
  - value：提供的属性值；

```
let counter = 100
let info = {
  name: "why",
  age: 10
}

provide("counter", counter)
provide("info", info)
```

Inject函数

- **在 后代组件 中可以通过 inject 来注入需要的属性和对应的值：**
  - 可以通过 inject 来注入需要的内容；
- **inject可以传入两个参数：**
  - 要 inject 的 property 的 name；
  - 默认值；

```
const counter = inject("counter")
const info = inject("info")
```

数据的响应式

- 为了增加 provide 值和 inject 值之间的响应性，我们可以在 provide 值时使用 ref 和 reactive。

```
let counter = ref(100)
let info = reactive({
  name: "why",
  age: 18
})
provide("counter", counter)
provide("info", info)
```

侦听数据的变化

- 在前面的Options API中，我们可以通过watch选项来侦听data或者props的数据变化，当数据变化时执行某一些操作。
- 在Composition API中，我们可以使用watchEffect和watch来完成响应式数据的侦听；
  - watchEffect：用于自动收集响应式数据的依赖；
  - watch：需要手动指定侦听的数据源；

Watch的使用

- **watch的API完全等同于组件watch选项的Property：**
  - watch需要侦听特定的数据源，并且执行其回调函数；
  - 默认情况下它是惰性的，只有当被侦听的源发生变化时才会执行回调；

```
const name = ref("kobe")

watch(name, (newValue, oldValue) => {
  console.log(newValue, oldValue);
})

const changeName = () => {
  name.value = "james";
}
```

侦听多个数据源

- 侦听器还可以使用数组同时侦听多个源：

```
const name = ref("why");
const age = ref(18)

const changeName = () => {
  name.value = "james";
}

watch([name, age], (newValues, oldValues) => {
  console.log(newValues, oldValues);
})
```

watch的选项

- 如果我们希望侦听一个深层的侦听，那么依然需要设置 deep 为true：
  - 也可以传入 immediate 立即执行；

```
const info = reactive({
  name: "why",
  age: 18,
  friend: {
    name: "kobe"
  }
})

watch(info, (newValue, oldValue) => {
  console.log(newValue, oldValue)
}, {
  immediate: true,
  deep: true
})
```

watchEffect

- **当侦听到某些响应式数据变化时，我们希望执行某些操作，这个时候可以使用 watchEffect。**
- 我们来看一个案例：
  - 首先，watchEffect传入的函数会被立即执行一次，并且在执行的过程中会收集依赖；
  - 其次，只有收集的依赖发生变化时，watchEffect传入的函数才会再次执行；

```
const name = ref("why");
const age = ref(18);

watchEffect(() => {
  console.log("watchEffect执行~", name.value, age.value);
})
```

watchEffect的停止侦听

- **如果在发生某些情况下，我们希望停止侦听，这个时候我们可以获取watchEffect的返回值函数，调用该函数即可。**
- 比如在上面的案例中，我们age达到20的时候就停止侦听：

```
const stopWatch = watchEffect(() => {
  console.log("watchEffect执行~", name.value, age.value);
});

const changeAge = () => {
  age.value++;
  if (age.value > 20) {
    stopWatch();
  }
};
```

useCounter

- 我们先来对之前的counter逻辑进行抽取：

```
import { ref } from 'vue'

export function useCounter() {
  const counter = ref(0);

  const increment = () => counter.value++
  const decrement = () => counter.value--

  return {
    counter,
    increment,
    decrement
  }
}
```

useTitle

- 我们编写一个修改title的Hook：

```
import { ref, watch } from 'vue'

export function useTitle(title = '默认值') {
  const titleRef = ref(title);

  watch(titleRef, (newValue) => {
    document.title = newValue;
  }, {
    immediate: true
  })

  return titleRef;
}
```

useScrollPosition（作业）

- 我们来完成一个监听界面滚动位置的Hook：

```
import { ref } from "vue";

export function useScrollPosition() {
  const scrollX = ref(0)
  const scrollY = ref(0)

  document.addEventListener('scroll', () => {
    scrollX.value = window.scrollX
    scrollY.value = window.scrollY
  })

  return { scrollX, scrollY }
}
```

script setup语法

- **<script setup> 是在单文件组件 (SFC) 中使用组合式 API 的编译时语法糖，当同时使用 SFC 与组合式 API 时则推荐该语法。**
  - 更少的样板内容，更简洁的代码；
  - 能够使用纯 Typescript 声明 prop 和抛出事件；
  - 更好的运行时性能；
  - 更好的 IDE 类型推断性能；
- **使用这个语法，需要将 setup attribute 添加到 <script> 代码块上：**

```
<script setup>

console.log("Hello World")

</script>
```

- 里面的代码会被编译成组件 setup() 函数的内容：
  - 这意味着与普通的 <script> 只在组件被首次引入的时候执行一次不同；
  - <script setup> 中的代码会在每次组件实例被创建的时候执行。

顶层的绑定会被暴露给模板

- 当使用 <script setup> 的时候，任何在 <script setup> 声明的顶层的绑定 (包括变量，函数声明，以及 import 引入的内容) 都能在模板中直接使用：

```
<script setup>
const message = "Hello World"
function btnClick() {
  console.log("btnClick")
}
</script>


<template>
  <h2>message: {{ message }}</h2>
  <button @click="btnClick">按钮</button>
</template>
```

- 响应式数据需要通过ref、reactive来创建。

导入的组件直接使用

- <script setup> 范围里的值也能被直接作为自定义组件的标签名使用：

```vue
<script setup>

import ShowInfo from './ShowInfo.vue'

</script>



<template>
  <show-info></show-info>
</template>
```

defineProps() 和 defineEmits()

- 为了在声明 props 和 emits 选项时获得完整的类型推断支持，我们可以使用 defineProps 和 defineEmits API，它们将自动地在 <script setup> 中可用：

```vue
<script setup>
  const props = defineProps({
    name: {
      type: String,
      default: ""
    },
    age: {
      type: Number,
      default: 0
    }
  })

  const emit = defineEmits(["changeAge"])
  function changeAge() {
    emit("changeAge", 200)
  }
</script>
```

```vue
<template>
  <h2>ShowInfo: {{ name }}-{{ age }}</h2>
  <button @click="changeAge">修改age</button>
</template>
```

defineExpose()

- **使用 <script setup> 的组件是默认关闭的：**
  - 通过模板 ref 或者 $parent 链获取到的组件的公开实例，不会暴露任何在 <script setup> 中声明的绑定；
- **通过 defineExpose 编译器宏来显式指定在 <script setup> 组件中要暴露出去的 property：**

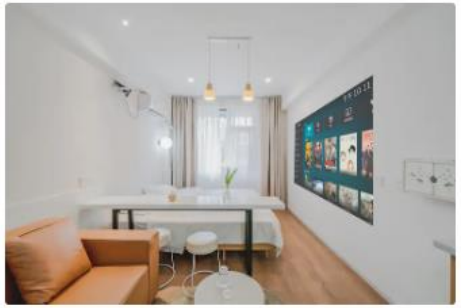
```
function foo() {
  console.log("foo function")
}

defineExpose({
  foo
})
```

```
const showInfoRef = ref(null)
function callShowInfo() {
  showInfoRef.value.foo()
}
```

案例实战练习

高分好评房源

品质房源，低至 5 折

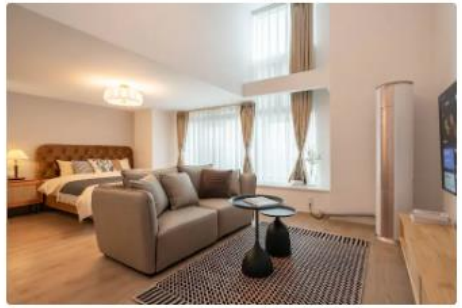


整套公寓型住宅 1室1卫1床

价格真实 实图拍摄 整套单独使用每客消毒 高清投影【方糖】人民北路地铁|龙湖上城|火车...

￥158/晚

超赞房东

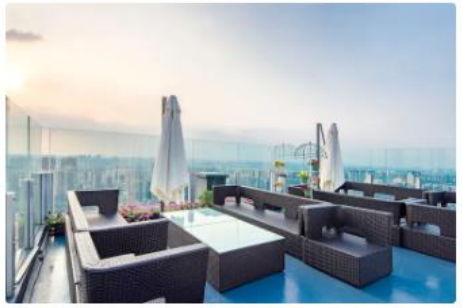


整套公寓型住宅 1室1卫1床

漫漫｜杨桃 轻奢一居室/地铁口/近春熙路太古里/4米9挑高/全景落地窗

￥201/晚

超赞房东



独立房间 1室1卫1床

【可月租】【网红美食街区】/近春熙路/宽窄巷子/熊猫基地//【轻奢大床房】

￥150/晚

超赞房东

整套公寓型住宅 1室1卫1床

可月租！品质大床！高空观景露台/步行地铁站/白天免费停车/直达春熙路/近建设巷小吃街

￥146/晚

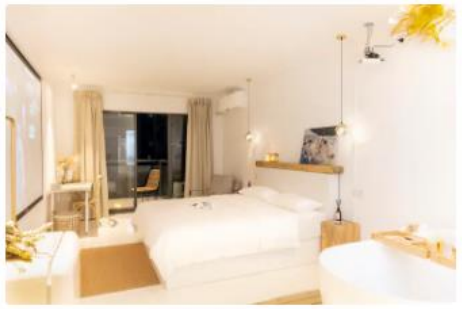
超赞房东

整套公寓型住宅 1室1卫1床

「精致mini房」楼下商场｜地铁直达近春熙路太古里｜建设巷小吃街｜白天免费停车｜可...

￥146/晚

超赞房东



整套公寓 1室1卫1床

【住.颜23】免清洁费/下楼就是太古里春熙路/高空浴缸/落地窗带阳台/百寸极米投影双地...

￥212/晚

超赞房东