

05 | 响应式开发操作：如何理解和使用Vue 3的响应式数据？

2022-11-30 杨文坚 来自北京



天下无鱼

<https://shikey.com/>

《Vue 3 企业级项目实战课》

[课程介绍 >](#)



讲述：杨文坚

时长 18:36 大小 17.05M



你好，我是杨文坚。

你应该经常在一些技术文章或者博客听到 Vue.js 的“响应式编程”，那究竟什么是“响应式”？

“响应式”这个词放在不同技术场景下有不同含义，Vue.js 一开始就是一个“响应式”的前端模板库，简单来讲，这个语境里的“响应式”就是“页面的模板内容能随着数据变化而重新渲染”。

Vue.js 最开始的响应式实现是基于 ES6 的一个 `Object.defineProperty` 的特性拦截监听数据变化，一旦监听到数据变化就触发对应依赖数据的模板重新渲染。

到了 Vue.js 的 3.x 版本，响应式就换成基于 ES6 的 **Proxy** 特性来实现的，Proxy 能监听一个对象的“读数据”和“写数据”的操作。最大的问题是因为 Vue.js 1.x 到 2.x 版本都是用的 `Object.defineProperty` 在监听数组变化时候，监听不到 `Array.push` 等数组变化操作，需要实现很多代码逻辑才能做好兼容。但用 **Proxy** 就能比较完美地直接监听数组的变化。

响应式开发是 **Vue.js** 框架的核心内容，开发者可以通过 **Vue.js** 的响应式的能力，直接用数据来驱动视图的变化，不需要写繁琐的 **DOM** 操作代码来处理视图的变化，可以让开发者能更加关注“如何设计数据来管理视图”，进而可以更加专注如何“根据业务逻辑来设计数据”，提升实现功能的开发效率。

那么既然 **Vue.js** 的响应式特性如此重要，学起来难不难呢？只要你跟着我的步骤由浅入深来进行学习，一节课下来就能掌握大致的使用方法啦。我现在就从响应式数据的基本操作来讲起。

响应式数据的基本操作


还记得我上节课一直演示的一个 **Vue.js** 的代码例子吗？这里再给你展示一下我们之前实现过的一个“计数器”，就是点击按钮，数字不断加 1 的计数器功能。

注意了，我这里用的是 **Vue.js** 推荐的组合式 **API** 的开发方式，**<script>** 标签需要加上 **setup** 属性。具体代码如下：

 复制代码

```
1 <template>
2   <div class="app">
3     <div class="text">Count: {{state.count}}</div>
4     <button class="btn" @click="onClick">Add</button>
5   </div>
6 </template>
7
8 <script setup>
9   import { reactive } from 'vue';
10  const state = reactive({
11    count: 0
12  });
13  const onClick = () => {
14    state.count ++;
15  }
16 </script>
```

在这段 **Vue.js 3** 的代码中，**reactive** 就是 **Vue.js 3** 官方提供的响应式 **API**，**state** 就是通过响应式 **API** 声明的响应式数据。**state** 这个数据可以直接在模板里使用，例如上述代码中我们使用了对象 **state** 里的 **count** 数据进行显示。

那么响应式体现在哪呢？你看，上述代码里其实还实现了一个点击事件的函数 `onClick`，这个函数实现了对 `state.count` 数值的自增操作。每当触发这个 `onClick` 事件，`state.count` 就会自动加 1，对应使用到 `state.count` 的模板也会随之重新渲染展示最新的数据内容。 <https://shikey.com/>

上述功能代码里的这种视图随着数据的变化，就是响应式的特征。基于 Vue.js 3 的响应式 API `reactive` 生成的数据，就是 Vue.js 3 的响应式数据。在 Vue.js 3 运行环境中，如果响应式数据的发生变化，那么依赖了数据的视图也会跟着变化。

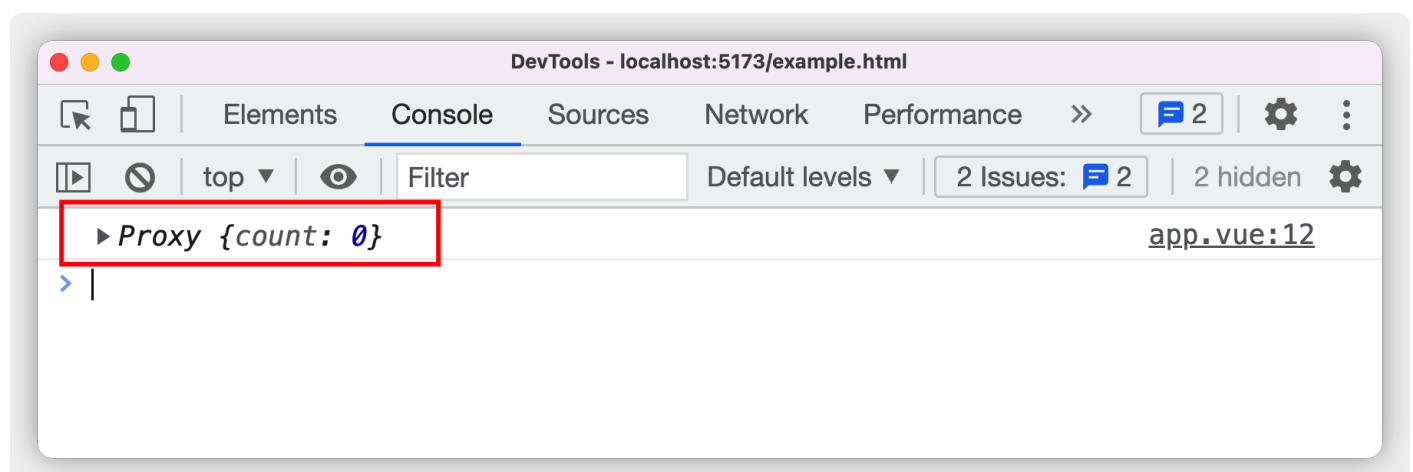
这里的响应式 API `reactive` 必须接受一个对象数据，通过内部封装返回一个 `Proxy` 类型的数据对象，这个 `Proxy` 数据就是响应式的核心。可以监听对象里每个属性的“读写”操作，通过“监听”或“劫持”属性的“读”和“写”的数据变化，触发依赖对应数据的模板视图进行重新渲染展示最新的数据值。

我们可以通过以下代码验证一下 `reactive` 返回的数据类型：

 复制代码

```
1 <template></template>
2
3 <script setup>
4 import { reactive } from 'vue';
5 const state = reactive({
6   count: 0
7 });
8
9 // 返回响应式数据内容
10 console.log(state)
11 </script>
```

执行这个 Vue.js 3 代码，在浏览器控制台打印的结果如下图所示：



我们可以看到，`reactive` 生成的响应式数据是一个 `Proxy` 数据。那么问题来了，什么是

`Proxy`？



`Proxy` 是 ES6 的一个新语法，功能与其英文字面意思一样，就是提供“代理”的功能，可以让对象数据实现属性的读和写操作的代理拦截处理，具体的使用方法你可以参考 [MDN 技术文档](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Proxy) 对 `Proxy` 的介绍。

通过上述内容，我们可以知道 `Vue.js 3` 的响应式 API `reactive` 是把对象数据转成 `Proxy` 类型数据进行数据的“代理”和“监听”，那么是不是所有响应式数据都必须转成 `Proxy` 数据类型呢？

不一定。其实我们将上述计数器的代码换成另外一个响应式 API，也能实现类似响应式功能的效果，代码如下所示：

复制代码

```
1 <template>
2   <div class="app">
3     <div class="text">Count: {{count}}</div>
4     <button class="btn" @click="onClick">Add</button>
5   </div>
6 </template>
7
8 <script setup>
9   import { ref } from 'vue';
10  const count = ref(0);
11  const onClick = () => {
12    count.value ++;
13  }
14 </script>
```

你可以看到，在这个修改过的 `Vue.js 3` 的代码里，`ref` 就是 `Vue.js 3` 官方提供的另一个响应式 API，`count` 就是通过响应式 API 声明的响应式数据。如果我们要修改这个 `count` 数据，就需要借助 `count.value` 属性来“间接”修改这个响应数据。但如果你只是想在模板中使用这个数据，就不需要执行 `value` 属性访问，直接使用这个数据名称就行了。

这里的 `ref` 可以接收一个基础数据类型或者对象数据类型，最后根据不同数据类型返回不同的响应式数据。

如果 `ref` 接收的是基础类型，例如 `Number`、`String`、`Boolean`、`Null`、`Undefined` 等，返回的响应式数据类型就是 `Vue.js 3` 定义的响应式数据类型 `RefImpl`。如果想在 `JavaScript` 读写基础类型的响应式数据，需要通过 `.value` 这个属性来进行操作，但是在模板 `template` 中就不需要这个 `value` 属性了。

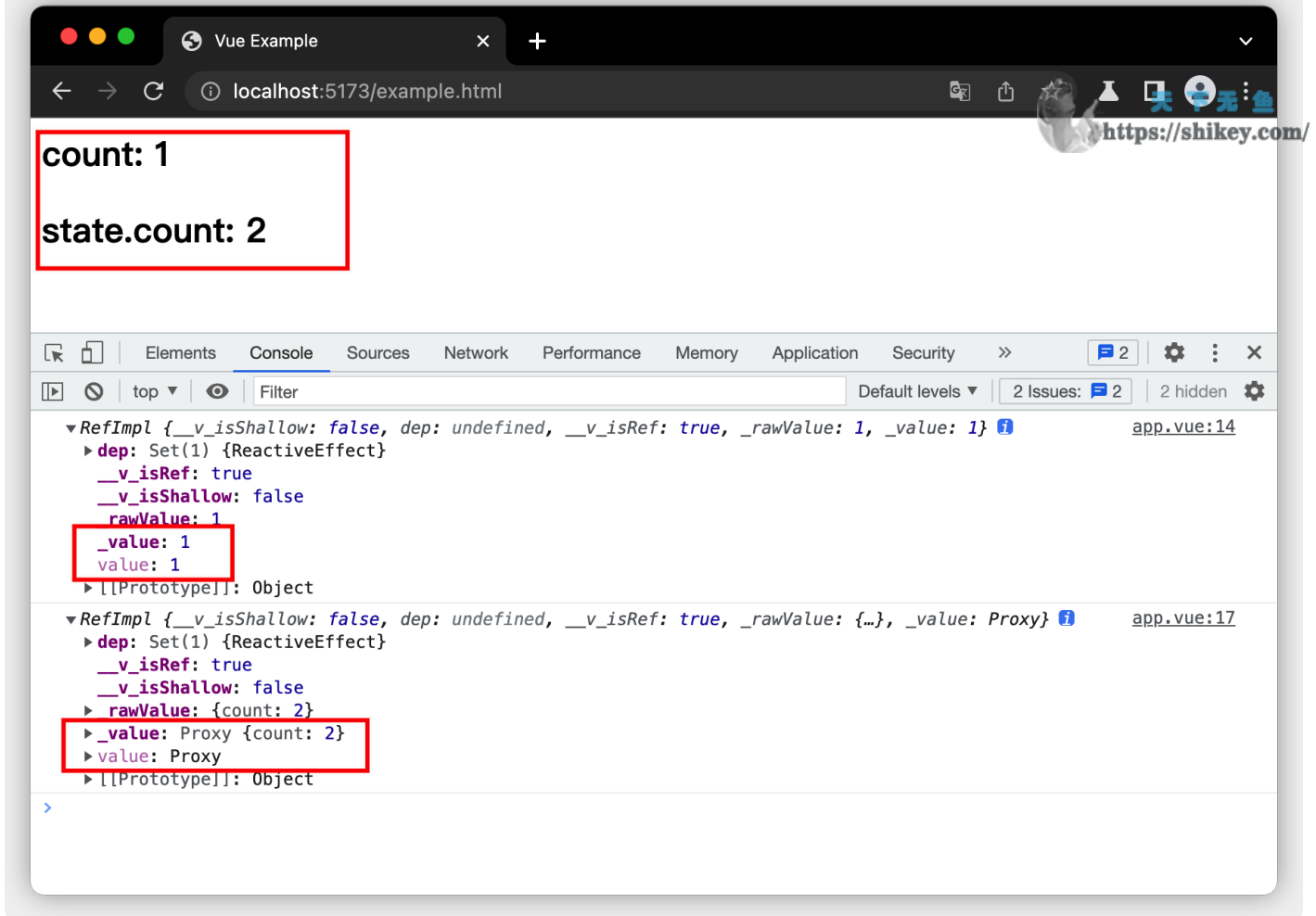
如果 `ref` 接收的是对象的数据类型，例如 `JSON`、`Array` 数据，那么返回的响应式数据类型也是 `RefImpl` 数据类型。基于 `Proxy` 封装的数据是放在 `RefImpl` 数据中的 `value` 属性里，如果你想在 `JavaScript` 操作这个对象的响应式数据，也是需要执行 `value` 属性，`value` 属性是用 `Proxy` 封装的对象数据，但是在模板 `template` 中就不需要这个 `value` 属性。

我们来做个代码实验，例如实现以下的代码片段，同样也是之前实现功能，是一个计数器操作。

 复制代码

```
1 <template>
2   <div>{{count}}</div>
3   <div>{{state.count}}</div>
4 </template>
5
6 <script setup>
7 import { ref } from 'vue';
8 const count = ref(1);
9 const state = ref({
10   count: 2
11 });
12
13 // ref 传入基本类型Number后 响应式数据内容
14 console.log(count)
15
16 // ref 传入对象数据后 响应式数据内容
17 console.log(state)
18 </script>
19
```

以上代码通过 `Vue.js 3` 编译在浏览器上运行后，在控制台打印结果如下图所示：



看到这里，你是不是有个疑问，为什么 `ref` 需要返回一个 `RefImpl` 类型，而不是直接返回 `Proxy` 类型呢？

这是因为 `Proxy` 只能代理监听对象类型的数据，例如 `JSON` 和 `Array` 数据，但是监听不了单纯的基础类型数据，例如 `Number`、`String` 之类。这个时候我们就需要设置一个 `value` 属性的对象来处理对基础类型数据的代理监听操作，下面举一个简单的例子：

复制代码

```
1 const data = {
2   _value: 1,
3   get value() {
4     console.log('我代理监听到 “读” Number数据操作，即将返回1')
5     return this._value;
6   },
7   set value(val) {
8     console.log('我代理监听到 “写” Number数据操作，即将设置新数据' + val)
9     this._value = val;
10  }
11 }
12
13 // 读数据时，会打印出监听的日志 '我代理监听到 “读” Number数据操作，即将返回1'
```

```
14 data.value;  
15 // 写数据时，会打印出监听的日志 '我代理监听到 "写" Number数据操作，即将设置新数据2'  
16 data.value = 2;
```



天下无鱼

<https://shikey.com/>

从上述例子可以看出，因为 Proxy 监听不了基础类型数据（Number、String、Boolean、Undefined、Null、BigInt 等这些基础类型），我们就需要构造出一个对象可以监听基础数据变化，实现响应式操作。

我们可以把基础类型数据（例如 Number 类型的数据）封装成一个对象，设置将基础类型读写操作转成对象属性 value 的 get 和 set 操作。这个时候，我们读写这个基础数据 value，就可以基于对象的 get 和 set 的内置方法来拦截监听数据变化，实现基础数据类型的响应式能力。

到了这里，你应该会发现，Vue.js 3 提供了两个重要的响应式 API，reactive 和 ref，他们都可以生成响应式数据，而且 ref 可以接收基础类型和对象类型数据，但是 reactive 只能接收对象类型数据。

那么，这里就有一个问题了，在项目开发中，我们应该如何选择这两种 API 呢？

ref 和 reactive 如何选择使用？

我们从上述内容可以知道，ref 和 reactive 最大的区别就是 ref 可以接收所有数据类型，根据不同数据类型返回对应的 RefImpl 响应式数据，但是操作数据需要访问 value 属性。而 reactive 只能接收对象数据类型，返回一个 Proxy 的响应式数据，可以访问数据属性来操作数据内容。

这个时候我们可以根据能力差异来做选择。**如果是要定义对象类型的响应式数据，那么我们可以优先选择 reactive 来定义**，这样子在 JavaScript 里可以直接操作对象响应式数据，就不需要多写一个 value 属性来操作了。而且，我们还可以避免对象本身带有 value 属性容易导致的理解上的属性混淆问题，例如下面代码所示：

复制代码

```
1 <template>  
2   <h2>data.value: {{refData.value}}</h2>  
3 </template>  
4  
5 <script setup>  
6 import { ref } from 'vue';  
7 // 如果有对象数据刚好是有value属性
```

```
8  const refData = ref({
9    value: 2,
10   count: 3
11 });
12
13 // 实际操作 value:2 数据需要这么操作
14 // 这里就会容易混淆 value 属性含义
15 refData.value.value += 1;
16
17 // 操作 count:3 这个数据
18 refData.value.count += 1;
19 </script>
```



从上述代码可以看出，对象数据还是少用 `ref` 来声明响应式数据，`value` 操作容易带来团队合作开发过程中理解和沟通成本。

如果要定义基础类型的响应式数据，就用 `ref` 来定义，通过访问基础类型的响应式数据的 `value` 属性来进行读写操作。

除了常用的 `reactive` 和 `ref` 之外，Vue.js 3 还有其他功能定义不同的响应式 API，你需要在特定场景下进行选择，相关的响应式 API 信息你可以查看 [Vue.js 3 的官方文档](https://v3.cn.cn.vuejs.org/zh/guide/reactivity-fundamentals.html)。

下面，我就根据企业中的开发习惯，结合官方对响应式 API 的建议，再挑几个响应式的 API 给你举例说明下它们的适用场景。

如何选择合适使用其它响应式 API?

我们先来讲讲“**去除响应式**”的响应式 API `toRaw`，也就是把响应式的关系解除。

平时在开发过程中，可能会将有些数据渲染在模板里的表单中，我们可以在表单中更改数据内容，但是又不希望表单里每次数据变化都触发响应式作用，这个时候就需要用到 `toRaw` 来解除掉数据的响应式。具体案例代码如下述源码所示：

复制代码

```
1 <template>
2   <form>
3     <input v-model="data.name" placeholder="用户名称" />
4     <br/>
5     <textarea v-model="data.info" placeholder="用户信息" />
6     <div>名称: {{data.name}}</div>
7     <div>信息: {{data.info}}</div>
8   </form>
```



```

9 </template>
10
11 <script setup>
12 import { reactive, toRaw } from 'vue';
13 import VUser from './user.vue';
14 const state = reactive({
15   name: '张三',
16   info: '某某公司前端开发工程师，有3年前端工作经验',
17 });
18 const data = toRaw(state);
19 </script>

```



上述代码中，我先基于 `reactive` 创建了一个响应式数据 `state`，里面存放了用户数据，再用 `toRaw` 进行解除掉响应式关系，返回一个原始的 JSON 数据 `data`，最后将其放在模板里的表单进行渲染。这样，当我们在表单的 `<input>` 和 `<textarea>` 组件中编辑这个原始数据 `data` 时，就不会触发响应式关系，也不会触发内容的重新渲染。

接下来我们要讲的**响应式 API watch**，这是监听响应式数据源变化的 API，也就是监听指定响应式数据变化，触发对应的回调函数，常用于处理数据变化的副作用操作，例如输入框经常需要用到监听数据变化做字数统计的操作、统计一些中文文字个数等，如下述代码所示：

复制代码

```

1 <template>
2   <form>
3     <textarea v-model="state.text" placeholder="信息" />
4     <div>中文字数: {{state.zhCount}}</div>
5   </form>
6 </template>
7
8 <script setup>
9 import { reactive, watch } from 'vue';
10
11 // 计算文本中文个数函数
12 function countZhText(txt) {
13   const zhRegExp = /[\\u4e00-\\u9fa5]{1,}/g;
14   const zhList = txt.match(zhRegExp);
15   let count = 0;
16   zhList.forEach((item) => {
17     count += item.length;
18   });
19   return count;
20 }
21 const defaultText = '今天是2022年01月01日'
22 const state = reactive({
23   text: defaultText,

```

```
24   zhCount: countZhText(defaultText),
25 });
26
27 watch(
28   // 监听 state.text 的变化
29   [() => state.text ],
30   ([ text ], [ prevText ]) => {
31     // 当监听到state.text 变化，就会触发这个回调函数
32     state.zhCount = countZhText(text);
33   }
34 )
35 </script>
```



上述代码运行后，`watch` 会监听 `state.text` 单独的数据变化，然后在 `<textarea>` 里编辑数据时，触发 `watch` 的回调函数，来计算 `state.text` 里的中文个数，计算完之后会修改 `state.zhCount` 单独触发视图显示最新的中文个数。更多 `watch` 的使用方法，你可以查看 [官方文档](#)。

通过我们前面对这两个响应式 API，以及 `reactive` 和 `ref` 这两个设置响应式数据的 API 的讲解，你应该能体会到，**响应式 API 基本上可以设置响应式数据、解除响应式状态和监听响应式数据这基本三种类型**。

那么我们在日常开发中，应该如何根据开发场景进行选择合适响应式 API 呢？

答案就是要先理清楚我们需要什么响应式操作，判断到底是设置、解除，还是监听响应式数据，然后在官方的 API 文档中 <https://cn.vuejs.org/api/> 找到自己合适的响应式处理场景。

不过，当你知道如何在合适场景中选择合适的响应式 API 进行开发的时候，并不能代表你已经掌握了 Vue.js 3 的响应式开发，你还需要知道在响应式开发中可能会遇到什么“坑”。那么，为了避免遇到这类“坑”，在做响应式开发时我们需要注意什么呢？

Vue.js 3 的响应式开发有什么需要注意的？

这里主要有下面这三个注意点：

- 响应式数据解构或者属性赋值后，可能会丢失响应式联系；
- 慎用浅层响应式作用 API；
- 慎用副作用 API。

第一个注意点就是“响应式数据解构或者属性赋值后，可能会丢失响应式联系”。怎么讲呢？我先给你展示一个代码例子：



复制代码

```
1 <template>
2   <textarea v-model="text" placeholder="文本信息" />
3   <div>文本信息: {{text}}</div>
4 </template>
5
6 <script setup>
7 import { reactive } from 'vue';
8 const state = reactive({
9   text: '今天是2022年01月01日',
10 });
11 const { text } = state;
12 </script>
```

你可以看到，上述代码中，`text` 的响应式联系并不会生效，`<textarea>` 修改 `text` 内容后，都不会触发页面的展示文本 `text` 的视图更新渲染。这是为什么呢？

我来一一分析一下。这里，我们用 `reactive` 定义了一个 `state` 响应式 JSON 数据，但是在之后又把其中的 `state.text` 解构赋值给了变量 `text`，这就会“断掉”了响应式的联系，导致再怎么更新 `text` 都不会触发视图重新更新渲染。

所以你在使用响应式数据时，要注意属性解构出来或者赋值出去，可能会带来“响应式联系的断开”，尽量避免相关的操作。

第二个注意点就是“慎用浅层响应式作用 API”，例如 `shallowReactive` 和 `shallowReadonly`。为什么呢？

因为 `shallowReactive` 这类响应式 API 生成的响应式对象数据，只作用对象数据的下一级属性，至于对象的下下一级属性就作用不到了。如果没什么特殊的需求，我们就尽量少用这类 API，这个 Vue.js 3 官方在 API 文档说明也提到过。

第三个注意点就是“慎用副作用 API”。为什么呢？我来把刚刚的一个代码例子改一下：

复制代码

```
1 <template>
```

```

2   <form>
3     <textarea v-model="state.text" placeholder="信息" />
4     <div>中文字数: {{state.zhCount}}</div>
5   </form>
6 </template>
7
8 <script setup>
9 import { reactive, watch } from 'vue';
10
11 // 计算文本中文个数函数
12 function countZhText(txt) {
13   const zhRegExp = /[\\u4e00-\\u9fa5]{1,}/g;
14   const zhList = txt.match(zhRegExp);
15   let count = 0;
16   zhList.forEach((item) => {
17     count += item.length;
18   });
19   return count;
20 }
21 const defaultText = '今天是2022年01月01日'
22 const state = reactive({
23   text: defaultText,
24   zhCount: countZhText(defaultText),
25 });
26
27 watch(
28   // 监听 state.text 的变化
29   [() => state.text, () => state.zhCount ],
30   ([ text ], [ prevText ]) => {
31     // 每当 state.text 变化，这个打印会触发两次
32     console.log('正在监听变化...')
33     // 当监听到state.text 变化，就会触发这个回调函数
34     state.zhCount = countZhText(text);
35   }
36 )
37 </script>

```



你看，上述代码中，每当 `state.text` 变化，打印代码 `console.log('正在监听变化...')` 就会触发两次，这是为什么呢？因为监听修改 `state.text` 变化，回调函数里会修改 `state.zhCount` 数据，但是 `state.zhCount` 也被 `watch` 函数监听，这个时候会再次触发回调函数，所以就会触发两次。

现在上述代码在执行过程中，只是修改一个数据，连锁反应触发两次回调。如果监听控制不好，可能会陷入监听回调函数的死循环执行，所以在使用监听副作用的 API 时候，要注意回调内部的操作和依赖之间的关系，尽可能谨慎使用。

总结

通过今天的讲解，你应该能理解 Vue.js 3 的响应式开发操作了，正是由于 Vue.js 的响应式特性，开发者才可以很方便地实现自己想要的页面功能。



我在此总结一下 Vue.js 3 的响应式开发操作的注意点，具体如下：

- 根据不同数据类型选择合适的响应式 API，例如基础数据用 `ref`，对象数据用 `reactive`；
- 如果想消除数据的响应式特性，可以通过 `toRaw` 来进行消除，将响应式数据变成普通数据；
- 不要随便解构响应式数据的属性，把属性赋值给其他变量的时候，赋值出去的数据容易“断开”响应式的联系；
- 监听响应式数据变化的其它副作用操作，可以通过 `watch` 来监听处理事件；
- 以上几点如果不能满足你的开发需要，去看官方的其它响应式 API，但是要慎用其它响应式 API。

为什么要“慎用”其它响应式 API 呢？这是因为凡事都有两面性的，Vue.js 3 提供了这么多的响应式能力，运用不当可能带来的不是方便，而是问题，甚至是故障。我在文稿里也提到一些响应式操作使用不恰当时，会带来的其他意想不到的问题。

所以在使用 Vue.js 3 的响应式 API 时候，要注意响应式的作用范围以及官方对个别 API 的“慎用提醒”，一般我们开发过程中尽可能做到“最小可用”原则就好，没必要用的技术特性或者 API 就尽量慎用或者少用，避免带来不必要的问题。

思考

我们这节课内容都是基于组合式 API（Composition API）的开发方式来进行响应式操作，那么如果换成组合式 API（Options API）的开发方式，响应式功能的实现应该怎么操作？

🔗 完整的代码在这里

分享给需要的人，Ta 购买本课程，你将得 18 元

📄 生成海报并分享



上一篇 04 | 模版语法和JSX语法：你知道Vue可以用JSX写吗？

下一篇 06 | 跨组件数据通信：常见的组件间数据通信方式有哪些？

精选留言 (4)

💬 写留言



风太太大大

2022-11-30 来自湖北

个人觉得文中有句话说的不是很合理，怕引起误会。“简单来讲，这个语境里的“响应式”就是“页面的模板内容能随着数据变化而重新渲染”。

这样是否合理，常规来看，这里这个语境里的“响应式”就是“页面的模板内容 及其他数据 能随着数据变化而重新渲染”。而实际响应式就是：当依赖的数据变化了，会更新使用这个数据相关的函数，内容模板是依赖数据试用的函数的一种，还有compute函数，watch函数等。

共 1 条评论 >



2



01

2022-12-01 来自福建

全文提到好几次JSON 数据。但是对象并不是JSON

共 1 条评论 >



都市夜归人 (酥)

2022-12-01 来自江苏

watch(

 // 监听 state.text 的变化

 [()] => state.text],

 ([text], [prevText]) => {

 // 当监听到state.text 变化，就会触发这个回调函数

 state.zhCount = countZhText(text);

 }

)

这段语法不太理解，请问为何要将被监听的变量放到数组里？谢谢！

共 1 条评论 >





风太大太大

2022-11-30 来自湖北

那么如果换成组合式 API（Options API）的开发方式，响应式功能的实现应该怎么操作？

1. 使用原vue2的写法，把响应的数据放在data函数的返回值中。只要后续直接修改这个值就是响应式。
2. 利用vue3的写法.使用setUp函数，照样可以使用reactive函数和ref函数。

共 1 条评论>

