浅谈：为啥vue和react都选择了Hooks?

📖阅读本文，你将:

1. 初步了解 Hooks 在 vue 与 react 的现状
2. 听一听本文作者关于 Hooks 的定义和总结
3. 弄懂"为什么我们需要 Hooks"
4. 进行一些简单的 Hooks 实践

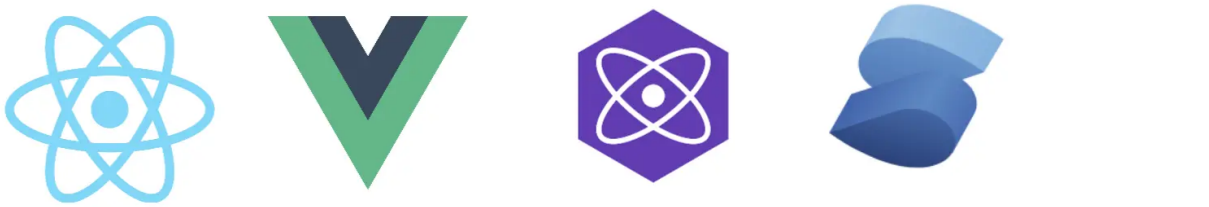
一、hooks: 什么叫大势所趋?

2019年年初， react 在 16.8.x 版本正式具备了 hooks 能力。

2019年6月，尤雨溪在 vue/github-issues 里提出了关于 vue3 Component API 的提案。（vue hooks的基础）

在后续的 react 和 vue3 相关版本中，相关 hooks 能力都开始被更多人所接受。

除此之外， solid.js 、 preact 等框架，也是开始选择加入 hooks 大家庭。



可以预见，虽然目前仍然是 class Component 和 hooks api 并驾齐驱的场面，但未来几年里， hooks 极有可能取代 class Component 成为业内真正的主流。

二、什么是 hooks?

> 年轻时你不懂我，就像后来我不懂 hooks 😭。

2.1 hooks 的定义

"hooks" 直译是 “钩子”，它并不仅是 react，甚至不仅是前端界的专用术语，而是整个行业所熟知的用语。通常指：

> 系统运行到某一时期时，会调用被注册到该时机的回调函数。

比较常见的钩子有：windows 系统的钩子能监听到系统的各种事件，浏览器提供的 onload 或 addEventListener 能注册在浏览器各种时机被调用的方法。

以上这些，都可以被称一声 "hook"。

但是很显然，在特定领域的特定话题下，hooks 这个词被赋予了一些特殊的含义。

在 react@16.x 之前，当我们谈论 hooks 时，我们可能谈论的是“组件的生命周期”。

但是现在，hooks 则有了全新的含义。

以 react 为例，hooks 是：

> 一系列以 “use” 作为开头的方法，它们提供了让你可以完全避开 class式写法，在函数式组件中完成生命周期、状态管理、逻辑复用等几乎全部组件开发工作的能力。

简化一下：

> 一系列方法，提供了在函数式组件中完成开发工作的能力。

（记住这个关键词：**函数式组件**）

js 复制代码

```
import { useState, useEffect, useCallback } from 'react';
// 比如以上这几个方法，就是最为典型的 Hooks
```

而在 vue 中， hooks 的定义可能更模糊，姑且总结一下：

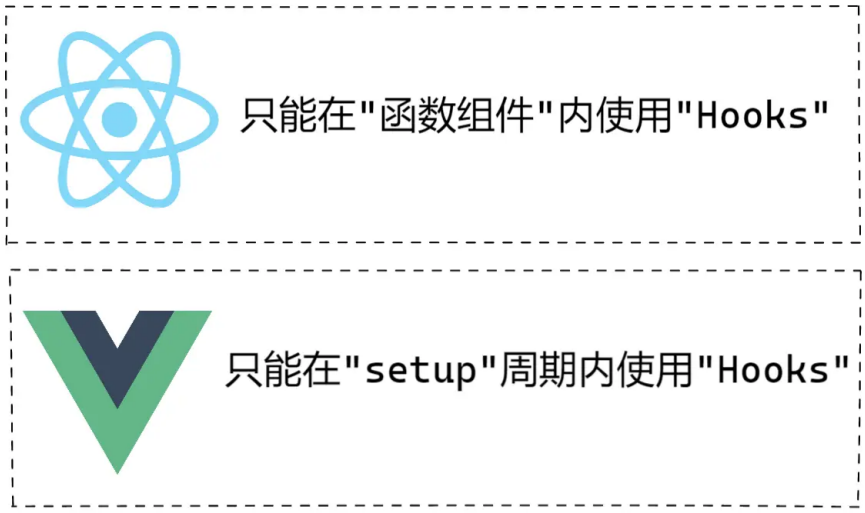
> 在 vue 组合式API里，以 “use” 作为开头的，一系列提供了组件复用、状态管理等开发能力的方法。

（关键词：**组合式API**）

```js
import { useSlots, useAttrs } from 'vue';
import { useRouter } from 'vue-router';
// 以上这些方法，也是 vue3 中相关的 Hook！
```

如： useSlots、 useAttrs、 useRouter 等。

但主观来说，我认为vue 组合式API其本身就是“vue hooks”的关键一环，起到了 react hooks里对生命周期、状态管理的核心作用。（如 onMounted、 ref 等等）。



如果按这个标准来看的话， vue 和 react 中 hooks 的定义，似乎都差不多。

那么为什么要提到是以 “use” 作为开头的方法呢?

2.2 命名规范 和 指导思想

通常来说， hooks 的命名都是以 use 作为开头，这个规范也包括了那么我们自定义的 hooks。

为什么?

因为 ~~(爱情-误)~~ 约定。

在 react 官方文档里，对 hooks 的定义和使用提出了**“一个假设、两个只在”**核心指导思想。~~(播音腔)~~

一个假设： 假设任何以 「use」 开头并紧跟着一个大写字母的函数就是一个 Hook。

第一个只在： 只在 React 函数组件中调用 Hook，而不在普通函数中调用 Hook。（Eslint 通过判断一个方法是不是大坨峰命名来判断它是否是 React 函数）

第二个只在： 只在最顶层使用 Hook，而不要在循环，条件或嵌套函数中调用 Hook。

因为是约定，因而 useXxx 的命名并非强制，你依然可以将你自定义的 hook 命名为 byXxx 或其他方式，但不推荐。

因为约定的力量在于：我们不用细看实现，也能通过命名来了解一个它是什么。

以上**“一个假设、两个只在”**总结自 react 官网：

1. zh-hans.reactjs.org/docs/hooks-...
2. zh-hans.reactjs.org/docs/hooks-...

三、为什么我们需要 hooks ？

3.1 更好的状态复用

> 怼的就是你， mixin ！

在 class 组件模式下，状态逻辑的复用是一件困难的事情。

假设有如下需求：

> 当组件实例创建时，需要创建一个 state 属性： name，并随机给此 name 属性附一个初始值。除此之外，还得提供一个 setName 方法。你可以在组件其他地方开销和修改此状态属性。

> 更重要的是: 这个逻辑要可以复用，在各种业务组件里复用这个逻辑。

在拥有 Hooks 之前，我首先会想到的解决方案一定是 mixin。

代码如下：（此示例采用 vue2 mixin 写法）

js 复制代码

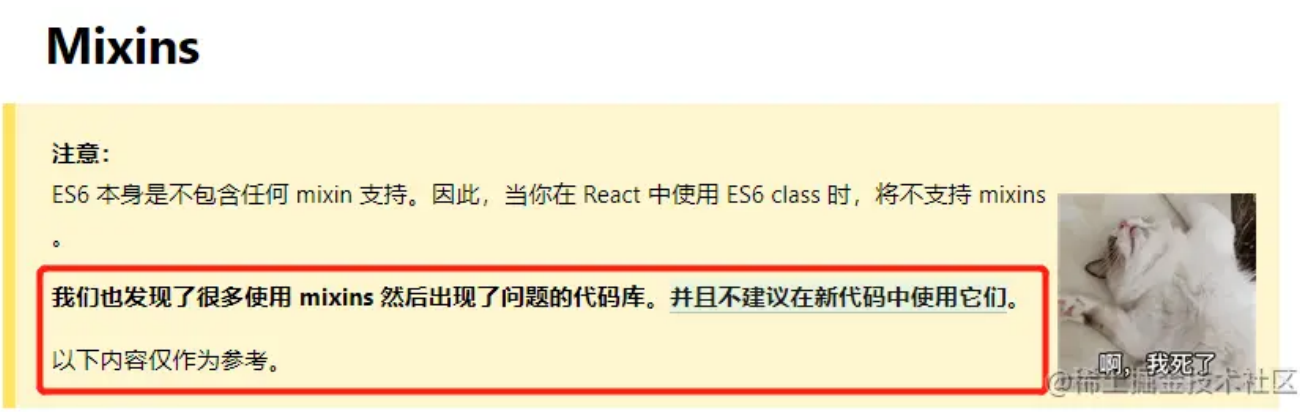
```
// 混入文件：name-mixin.js
export default {
  data() {
    return {
      name: genRandomName() // 假装它能生成随机的名字
    }
  },
  methods: {
    setName(name) {
      this.name = name
    }
  }
}
```

jsx 复制代码

```
// 组件：my-component.vue
<template>
  <div>{{ name }}</div>
<template>
```

```
<script>
import nameMixin from './name-mixin';
export default {
  mixins: [nameMixin],
  // 通过mixins, 你可以直接获得 nameMixin 中所定义的状态、方法、生命周期中的事件等
  mounted() {
    setTimeout(() => {
      this.setName('Tom')
    }, 3000)
  }
}
<script>
```

粗略看来， mixins 似乎提供了非常不错的复用能力，但是，react官方文档直接表明：



为什么呢?

因为 mixins 虽然提供了这种状态复用的能力，但它的**弊端实在太多了**。

弊端一：难以追溯的方法与属性!

试想一下，如果出现这种代码，你是否会怀疑人生：

```
export default {
  mixins: [ a, b, c, d, e, f, g ], // 当然，这只是表示它混入了很多能力
  mounted() {
    console.log(this.name)
    // mmp!这个 this.name 来自于谁？我难道要一个个混入看实现?
```

```
  }
}
```

又或者：

```
a.js mixins: [b.js]

b.js mixins: [c.js]

c.js mixins: [d.js]

// 你猜猜看, this.name 来自于谁?
// 求求你别再说了，我血压已经上来了
```

弊端二：覆盖、同名？贵圈真乱！

当我同时想混入 mixin-a.js 和 mixin-b.js 以同时获得它们能力的时候，不幸的事情发生了：

由于这两个 mixin 功能的开发者惺惺相惜，它们都定义了 this.name 作为属性。

这种时候，你会深深怀疑， mixins 究竟是不是一种科学的复用方式。

弊端三：梅开二度？代价很大！

仍然说上面的例子，如果我的需求发生了改变，我需要的不再是一个简单的状态 name，而是分别需要 firstName 和 lastName。
此时 name-mixin.js 混入的能力就会非常尴尬，因为我无法两次 mixins 同一个文件。

当然，也是有解决方案的，如：

```
// 动态生成mixin
function genNameMixin(key, funcKey) {
  return {
    data() {
```

```
      return {
        [key]: genRandomName()
      }
    },
    methods: {
      [funcKey]: function(v) {
        this.[key] = v
      }
    }
  }
}

export default {
  mixins: [
    genNameMixin('firstName', 'setFirstName'),
    genNameMixin('lastName', 'setLastName'),
  ]
}
```

确实通过动态生成 mixin 完成了能力的复用，但这样一来，无疑更加地增大了程序的复杂性，降低了可读性。

因此，一种新的 “状态逻辑复用” 就变得极为迫切了——它就是 Hooks！

Hook 的状态复用写法：

js 复制代码

```
// 单个name的写法
const { name, setName } = useName();

// 梅开二度的写法
const { name : firstName, setName : setFirstName } = useName();

const { name : secondName, setName : setSecondName } = useName();
```

相比于 mixins，它们简直太棒了！

1. 方法和属性好追溯吗？这可太好了，谁产生的，哪儿来的一目了然。
2. 会有重名、覆盖问题吗？完全没有！内部的变量在闭包内，返回的变量支持定义别名。
3. 多次使用，没开N度？你看上面的代码块内不就“梅开三度”了吗？

就冲**“状态逻辑复用”**这个理由，Hooks 就已经香得我口水直流了。

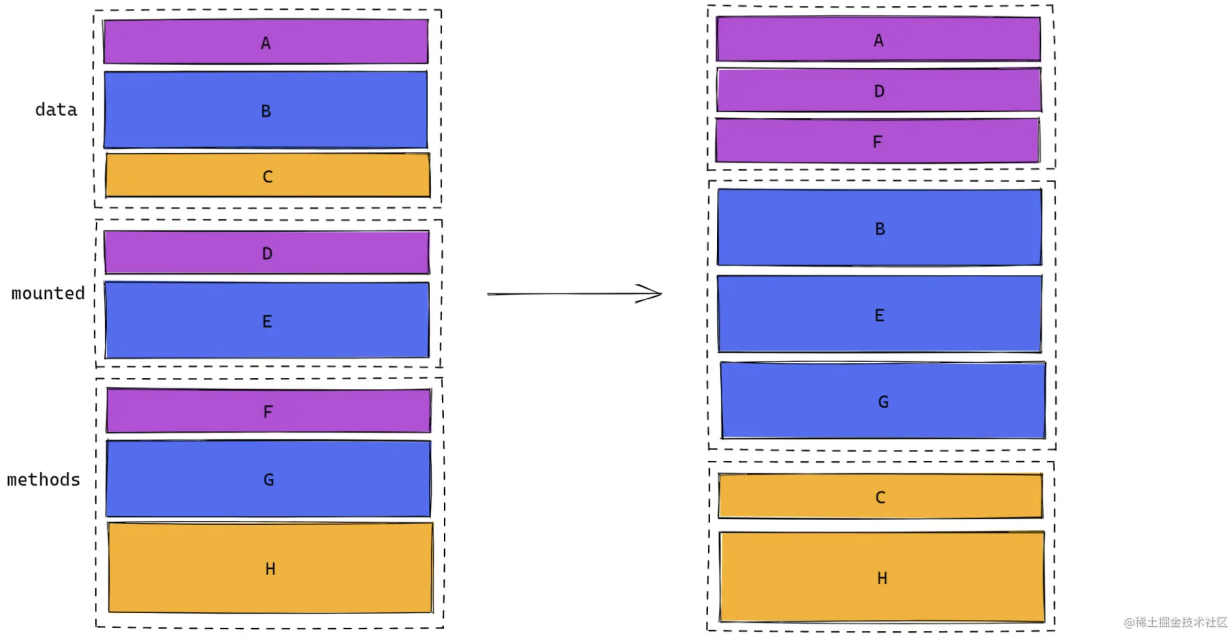
3.2 代码组织

> 熵减，宇宙哲学到编码哲学。

项目、模块、页面、功能，如何高效而清晰地组织代码，这一个看似简单的命题就算写几本书也无法完全说清楚。

但一个页面中，N件事情的代码在一个组件内互相纠缠确实是在 Hooks 出现之前非常常见的一种状态。

那么 Hooks 写法在代码组织上究竟能带来怎样的提升呢？



（假设上图中每一种颜色就代码一种高度相关的业务逻辑）

无论是 vue 还是 react，通过 Hooks 写法都能做到，将“分散在各种声明周期里的代码块”，通过 Hooks 的方式将相关的内容聚合到一起。

这样带来的好处是显而易见的：**“高度聚合，可阅读性提升”**。伴随而来的便是**“效率提升，bug变少”**。

按照“物理学”里的理论来说，这种代码组织方式，就算是“熵减”了。

3.3 比 class 组件更容易理解

> 尤其是 this 。

在 react 的 class 写法中，随处可见各种各样的 .bind(this)。（甚至官方文档里也有专门的章节描述了“为什么绑定是必要的？”这一问题）

vue 玩家别笑， computed: { a: () => { this } } 里的 this 也是 undefined。

很显然，绑定虽然“必要”，但并不是“优点”，反而是“故障高发”地段。

但在Hooks 写法中，你就完全不必担心 this 的问题了。

因为：

> 本来无一物，何处惹尘埃。

Hooks 写法直接告别了 this，从“函数”来，到“函数”去。

妈妈再也不用担心我忘记写 bind 了。

3.4 友好的渐进式

> 随风潜入夜，润物细无声。

渐进式的含义是：你可以一点点深入使用。

无论是 vue 还是 react，都只是提供了 Hooks API，并将它们的优劣利弊摆在了那里。并没有通过无法接受的 break change 来强迫你必须使用 Hooks 去改写之前的 class 组件。

你依然可以在项目里一边写 class 组件，一边写 Hooks 组件，在项目的演进和开发过程中，这是一件没有痛感，却悄无声息改变着一切的事情。

但是事情发展的趋势却很明显，越来越多的人加入了 Hooks 和 组合式API 的大军。

四、如何开始玩 hooks ？

4.1 环境和版本

在 react 项目中， react 的版本需要高于 16.8.0。

而在 vue 项目中， vue3.x 是最好的选择，但 vue2.6+ 配合 @vue/composition-api，也可以开始享受"组合式API"的快乐。

4.2 react 的 Hooks 写法

因为 react Hooks 仅支持"函数式"组件，因此需要创建一个函数式组件 my-component.js。

jsx 复制代码

```jsx
// my-component.js
import { useState, useEffect } from 'React'

export default () => {
 // 通过 useState 可以创建一个 状态属性 和一个赋值方法
 const [ name, setName ] = useState('')

 // 通过 useEffect 可以对副作用进行处理
 useEffect(() => {
  console.log(name)
 }, [ name ])

 // 通过 useMemo 能生成一个依赖 name 的变量 message
 const message = useMemo(() => {
  return `hello, my name is ${name}`
```

```
  }, [name])

  return <div>{ message }</div>
}
```

细节可参考 react 官方网站：react.docschina.org/docs/hooks-...

4.3 vue 的 Hooks 写法

vue 的 Hooks 写法依赖于 组合式API，因此本例采用 <script setup> 来写：

jsx 复制代码

```
<template>
  <div>
    {{ message }}
  </div>
</template>
<script setup>
import { computed, ref } from 'vue'
// 定义了一个 ref 对象
const name = ref('')
// 定义了一个依赖 name.value 的计算属性
const message = computed(() => {
  return `hello, my name is ${name.value}`
})
</script>
```

很明显， vue 组合式API里完成 useState 和 useMemo 相关工作的 API 并没有通过 useXxx 来命名，而是遵从了 Vue 一脉相承而来的 ref 和 computed。

虽然不符合 react Hook 定义的 Hook 约定，但 vue 的 api 不按照 react 的约定好像也并没有什么不妥。

参考网址：v3.cn.vuejs.org/api/composi...

五、开始第一个自定义 hook

除了官方提供的 Hooks Api，Hooks 的另外一个重要特质，就是可以自己进行“自定义 Hooks” 的定义，从而完成状态逻辑的复用。

开源社区也都有很多不错的基于 Hooks 的封装，比如 ahooks（ahooks.js.org/zh-CN/），又比如 vueuse （vueuse.org/）

我还专门写过一篇小文章介绍 vuehook：【一库】vueuse:我不许身为vuer,你的工具集只有lodash!。

那么，我们应该怎么开始撰写 “自定义Hooks” 呢？往下看吧！

5.1 react 玩家看这里👈

react 官方网站就专门有一个章节讲述“自定义Hook”。
（react.docschina.org/docs/hooks-...）

这里，我们扔用文章开头那个 useName 的需求为例，希望达到效果：

```js
const { name, setName } = useName();
// 随机生成一个状态属性 name，它有一个随机名作为初始值
// 并且提供了一个可随时更新该值的方法 setName
```

如果我们要实现上面效果，我们该怎么写代码呢？

```js
import React from 'react';

export const useName = () => {
```

```
  // 这个 useMemo 很关键
  const randomName = React.useMemo(() => genRandomName(), []);
  const [ name, setName ] = React.useState(randomName)

  return {
    name,
    setName
  }
}
```

忍不住要再次感叹一次，和 mixins 相比，它不仅使用起来更棒，就连定义起来也那么简单。

可能有朋友会好奇，为什么不直接这样写：

```
const [ name, setName ] = React.useState(genRandomName())
```

因为这样写是不对的，每次使用该 Hook 的函数组件被渲染一次时， genRandom() 方法就会被执行一次，虽然不影响 name 的值，但存在性能消耗，甚至产生其他 bug。

为此，我写了一个能复现错误的demo，有兴趣的朋友可以点开验证：
codesandbox.io/s/long-cher...

> 2022-02-03日补充更正：经掘友提醒，可以通过 React.useState(() => randomName()) 传参来避免重复执行，这样就不需要 useMemo 了，感谢！

5.2 vue 玩家看这里👈

vue3 官网没有关于 自定义Hook 的玩法介绍，但实践起来也并不困难。

目标也定位实现一个 useName 方法：

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

```
export const useName = () => {
  const name = ref(genRandomName())
  const setName = (v) => {
    name.value = v
  }
  return {
    name,
    setName
  }
}
```

5.3 vue 和 react 自定义 Hook 的异同