Vue有20样写法，你知道么？

引入

打开 Vue3 的官方文档，它首先会告诉你，Vue 的组件可以按两种不同的风格书写：选项式 API 和组合式 API。文档为我们提供一系列两种风格的代码参考，供我们按照偏好进行选择。

实际上，Vue3 组件可不止两种写法，而是多达十几种！然而，不管是什么写法，它们都是基于同一个底层系统实现的，概念之间也是彼此相通的，只是使用的接口不同。在实际开发中，我们也不会同时使用到那么多种写法，但是这并不意味着我们不需要去了解这些写法！

如果你仔细阅读 Vue3 的文档，会发现一些示例或者 api 看起来模棱两可，不知道这些 api 到底有什么用，或者阅读 Vue 的源码时，总是能发现一些对于我们来说意图不明的逻辑，那么，你可能先要了解一些 Vue 的写法。先了解一个东西怎么用，再去分析它是怎么实现的。

看完本文章，你会收获到：

1. vue 的渲染原理。
2. 什么是 defineComponent、h、creatVnode。
3. 渲染函数和 jsx 的区别。
4. 马上能够上手 jsx。
5. 轻松阅读 vue 文档的所有示例和 api。
6. 轻松看懂不同的 vue 组件写法。
7. 不管新手老手，都会对 Vue 有所新的认识。
8. 等等...

本文章遵从循序渐进的写作顺序，从易到难，轻松上手！

setup 语法糖

setup 语法糖应该是最常用的写法了。在 Vue3 中，我们想封装一个组件，最习惯的做法还是新建一个 Vue 文件，并将组件代码写在文件中。具体是：页面结构写在 template 中，页面逻辑写在 script 中，页面样式写在 style 中。

总之，我们将与该组件相关的代码都写在一起、放在一个文件中单独维护，在需要该组件的地方引入使用。

这里我们使用了 setup 语法糖，直接在 script 中书写我们的 setup 内部的逻辑。

```
1 <template>
2   <div>{{ name }}</div>
```

```
</template>

<script setup lang="ts">
import { ref } from "vue";
const name = ref("天气好");
</script>

<style scoped></style>
```

在 App. vue 中引入并使用：

```
// App.vue
<template>
  <User />
</template>

<script setup lang="ts">
import User from "./User.vue";
</script>

<style scoped></style>
```

注：后续写法尽管形式不同，但它们最终的目的都是导出一个组件，所以对于组件使用方来说（这里是 App. vue），怎么使用这个组件的代码都是不变的，所以将不再重复此代码。

Vue2 选项式写法

Vue2 经典写法

这种写法也是比较经典的。和 setup 语法糖写法类似。我们需要新建一个 vue 文件来存储我们的组件代码，然后在需要使用该组件的地方对其进行引入。区别在于，我们需要在 script 中导出一个 Vue 实例。

这里我们导出的其实是一个普通对象，该对象包含 data、methods 等属性。这个对象的属性都是可选的，即 option，翻译回来即“选项”。

```
<template>
  <div>{{ name }}</div>
</template>

<script lang="ts">
export default {
  data: () => {
```

```
    return {
      name: "天气好",
    };
  },
};
</script>

<style></style>
```

defineComponent 辅助函数

尽管我们在 script 语言块中导出的默认对象会被 vue 编译器当成 vue 实例，但不管怎么看，它依旧只是一个 plain object。在定义组件实例方面，vue 提供了一个名为 defineComponent 辅助接口。

```
<template>
  <div>{{ name }}</div>
</template>

<script lang="ts">
import { defineComponent } from "vue";
export default defineComponent({
  data: () => {
    return {
      name: "天气好",
    };
  },
});
</script>

<style></style>
```

尽管这个接口也不能改变我们导出的是一个普通对象的事实，但是它可以为我们的实例提供强大的类型推导。我们可以把它看成是一个返回 vue 实例的工厂函数，让我们的代码看起来更加规范。

Vue3 选项式写法

在 Vue3 中，官方引入了新的选项 setup，这是 Vue3 选项式写法和 Vue2 写法的主要区别。setup 选项的意义在于它允许我们在选项式的写法中引用和使用组合式的 api，比如 onMounted、ref、reactive 等。但对于我们来说，它对于我们有益的地方还是基于它封装起来的 setup 语法糖用起来很方便。

```
<template>
  <div>{{ name }}</div>
</template>
```

```
4
5 <script lang="ts">
6 export default {
7   setup() {
8     return {
9       name: "天气好",
10    };
11  },
12 };
13 </script>
14
15 <style></style>
```

使用 defineComponent 时，它能够提示我们 setup 将会接收到什么参数：

```
1 <template>
2   <div>{{ name }}</div>
3 </template>
4
5 <script lang="ts">
6 import { defineComponent } from "vue";
7 export default defineComponent({
8   setup(prop,context) {
9     return {
10      name: "天气好",
11    };
12  },
13 });
14 </script>
15
16 <style></style>
```

以上写法我们都是在 template 上书写我们的页面结构，这也是最常见的几种写法，下面我们来介绍几种了解 vue 底层必不可少的写法，渲染函数。

手写渲染函数

template 模板语法本质上也可以算是一种语法糖。在 vue 编译器上，template 中的内容最终会被翻译为渲染函数，挂载到 vue 实例的 render 属性上。当需要渲染组件时，vue 就执行一次 render，得到对应的虚拟节点树，最后再转变为真实 dom。

Vue 允许我们脱离 template，直接自己书写渲染函数。位置就在导出实例的 render 选项上：

```vue
<script lang="ts">
import { defineComponent } from "vue";
export default defineComponent({
  data: () => ({ name: "天气好" }),
  render() {
    return this.name;
  },
});
</script>

<style></style>
```

在 template 中，我们使用类似 html 的模板语法来描述我们的视图，在 render 函数中又如何描述呢？vue 提供了两个 api：createVnode 和 h。二者没有区别，h 函数只是 createVnode 的缩写。有了 render 函数，我们就不需要写 template 了。

```vue
<script lang="ts">
import { defineComponent, h } from "vue";
export default defineComponent({
  data: () => ({ name: "天气好" }),
  render() {
    return h("div", this.name);
  },
});
</script>

<style></style>
```

在上面的示例中，我们使用 h 函数生成了一个 vNode，并 return 出去，作为本组件最终在被使用时渲染出来的效果。

在 template 中我们可以使用 v-if、v-for、slot 等模板语法，在 h 函数中这些概念也是支持的，只是形式不同，这方面官方文档有具体的示例。总之，template 模板和 render 选项是可以相互替代的。

setup 返回渲染函数

setup 返回 render 方法

一般来说，在选项式语法中，setup 方法返回一个对象，该对象暴露给 template，供 template 使用，具体参考第三个例子（vue3 选项式写法）。如果我们不使用 template，也就没有返回对象的必要了。

在 Vue3 中，还有另外一种不使用 template 的写法，就是在 setup 方法中返回一个 render 方法。

```vue
<script lang="ts">
import { defineComponent, h, ref } from "vue";
export default defineComponent({
  setup() {
    const name = ref("天气好");
    return () => h("div", name.value);
  },
});
</script>

<style></style>
```

注意：

1. 在选项式中使用 setup 之后，一般不应该再使用 data、生命周期等在选项式写法中常用的选项，而应该把主要逻辑都写在 setup 中，并适当引入组合式的 api。比如，使用 ref，而不是 data 选项。
2. ref 自动解包是 template 特有的功能，h 函数是没有这个功能的。在 h 函数中引入 ref，记得理所当然地带上 `.value`。

defineComponent 传入setup

就注意中的第一点，我们可以采用下面这种写法：直接在 defineComponent 中书写 setup 函数（如果再省一点就是 setup 语法糖的写法了）。

```vue
<script lang="ts">
import { defineComponent, h, ref } from "vue";
export default defineComponent(() => {
  const name = ref("天气好");
  return () => h("div", name.value);
});
</script>

<style></style>
```

以上就是渲染函数的写法，是不是有点感觉了呢，一下子就学会了两个 api！后面会提到的 Jsx 写法其实也应该归为渲染函数写法的一种（只要不是 template，而是用 JavaScript 表达页面结构的，都是渲染函数），但是相对于 h 函数，jsx 并不是纯粹的 js，所以我将它们分成了两类。

Vue & Jsx

在render 中使用 jsx

有了前面两类写法介绍的铺垫，接下来引入 jsx 语法就没有什么难理解的点了。

jsx 在 vue 文件中是这样写的。在 render 渲染函数返回值处书写 jsx 替代 h 函数。书写纯 JavaScript 的 h 函数描述结构还是比较繁冗的，jsx 就是简化了的h 函数写法。

```
<script lang="tsx">
import { defineComponent } from "vue";
export default defineComponent({
  data() {
    return { name: "天气好" };
  },
  render() {
    return (
      <>
        <div>{this.name}</div>
      </>
    );
  },
});
</script>

<style></style>
```

在 setup 中使用jsx

jsx 和 setup 配合食用更加。在选项式风格中使用 setup，在 setup 中使用组合式 api，并且返回 jsx 书写的渲染函数。

```
<script lang="tsx">
import { defineComponent, ref } from "vue";
export default defineComponent({
  setup() {
    const name = ref("天气好");
    return () => <>{name.value}</>;
  },
});
</script>

<style></style>
```

defineComponent 简写

这个其实就是前面介绍过的 **defineComponent 传入 setup** 函数写法：这里的区别只是使用 jsx 替代了 h 函数。

```
1 html
2 复制代码
3 <script lang="tsx"import { defineComponent, ref } from "vue";export default
  defineComponent(() => {  const name = ref("天气好");  return () => (    <>
  <div{name.value}</div    </>  );});</script<style</style
```

自行导出 vNode 对象

我们也可以自己将 render 函数执行一遍，然后将得到的 jsx Element 导出，和上一个示例**defineComponent 简写**是十分相似。但是这段代码的缺点非常致命，它不支持接收外部传递来的属性参数。

```tsx
<script lang="tsx">
import { defineComponent, ref } from "vue";
export default defineComponent(() => {
  const name = ref("天气好");
  return () => (
    <>
      <div>{name.value}</div>
    </>
  );
});
</script>

<style></style>
```

不要使用这种写法。这里会提到这样写，只是因为和后面的**函数式组件（其二）** 写法有关联。本写法与其它写法都不同，其它写法导出的都是 JavaScript 对象或者 jsx 对象，而这里我们则是自己执行了一遍渲染函数并得到了虚拟节点，直接将虚拟节点导出去。既然都已经把虚拟节点创建出来了，那自然无法接收 props。

defineComponent 的第二个参数

如果 defineComponent 的第一个参数是 setup 函数，那么它的第二个参数则可以为组件的定义添加需要的选项，但一般除了补充 props 选项，不会再需要其它选项了（组合式 api 和 setup 的入参可以完全替代其它选项）。

```tsx
<script lang="tsx">
import { defineComponent } from "vue";
export default defineComponent(
  (props) => {
    return () => (
```

```
      <>
        <div>{props.userName}</div>
      </>
    );
  },
  {
    props: { userName: String },
  }
);
</script>

<style></style>
```

直接在 vue 中使用 jsx

这里 jsx 不再只作为返回值，而是直接被某处使用。它可以是被直接导出，或者用在 template 上。

直接导出 jsx 对象

直接将 jsx 对象导出使用。比前面的写法更简洁，做法就是把 setup 里面的内容提到外面。这里需要注意的是我们导出的是一段直接的 jsx 对象（jsx Element），而不是渲染函数。

```
<script lang="tsx">
import { ref } from "vue";
const name = ref("天气好");
const User = <>{name.value}</>;
export default User;
</script>

<style scoped></style>
```

直接用在 template 上

这种写法可以帮助你在自身的组件内复用一些颗粒度更小的组件，它和 setup 语法糖的写法非常接近，只是 User 变量可以作为标签直接使用。

```
<template>
  <User />
</template>

<script setup lang="tsx">
import { ref } from "vue";
const name = ref("天气好");
```

```
8 const User = <>{name.value}</>;
9 </script>
10
11 <style></style>
```

函数式组件（其一）

你还可以将 User 写成函数式组件，在本页面内使用。但它不会将连字符属性转换为小驼峰写法。这和**直接用在 template 上**的内容都是一样的，它们都是为了方便在组件本身复用一些常用的组件。

```
1 <template>
2   <User :user-name="name" />
3 </template>
4
5 <script setup lang="tsx">
6 import { ref } from "vue";
7 const name = ref("天气好");
8 const User = (props: { "user-name": string }) => {
9   return <>{props["user-name"]}</>;
10 };
11 </script>
12
13 <style></style>
```

如果你经常使用 tailwind，你可能就会知道什么情况下会出现小颗粒度的可复用标签，比如，一个加了一大堆类名的 div 标签。

独立的 Jsx 文件

以上介绍的所有写法，都是在 `.vue` 文件中书写的，而且也离不开 `<script lang="tsx">`。接下来的写法可以让我们脱离 Vue 的语法块框架，书写更像 jsx 的 jsx。

jsx 定义组件

我们需要新建一个 jsx/tsx 文件，然后只要保证导出的仍然是一个组件就可以了。有了前面的铺垫，我们不难发现，这不就是去掉 script 标签的选项式写法吗？确实！这是因为我故意在前面安排了选项式写法的例子，所以过渡到这里完全没有压力！

```
1 // User.tsx
2 import { ref } from 'vue'
3 export default {
4   setup() {
5     const name = ref('天气好')
```

```
    return () => <><div>{name.value}</div></>
  }
}
```

我还是推荐套上 defineComponent：

```
// User.tsx
import { ref, defineComponent } from 'vue'
export default defineComponent({
  setup() {
    const name = ref('天气好')
    return () => (<><div>{name.value}</div></>)
  }
});
```

同样地，前面对于 defineComponent 不同方式的使用这里也都可以的。比如导出普通对象并在 render 或者 setup 中使用 jsx 等等。从 vue 到 jsx，区别只是省下了 script 语法块。

vue2 选项式写法+jsx。

```
// User.tsx
import { defineComponent } from 'vue'
export default defineComponent({
  data: () => ({ name: '天气好' }),
  render() {
    return <><div>{this.name}</div></>
  }
});
```

导出普通对象：

```
// User.tsx
export default {
  data: () => ({ name: '天气好' }),
  render() {
    return <><div>{this.name}</div></>
  }
});
```

函数式组件（其二）

Vue 中支持的最像函数式组件的写法。

```tsx
// User.tsx
import { ref } from 'vue'

export default function User(props) {
  const name = ref('天气好')
  return <><div>{name.value}</div></>
}
```

该例和前面的**自行导出 vNode 对象**非常接近，这也是为什么即使后者存在不能接收参数的缺陷我也会提出来，因为二者都是使用接近函数式组件的写法来描述组件的，但是在 vue 文件中并没有办法直接导出这个函数组件，而是需要自行执行得到vNode。而在 jsx 文件中却可以将其导出，并且支持接收参数。

如果你需要为其定义 props，也不需要使用 **defineComponent 的第二个参数**为你提供什么 props 选项，而是直接在函数式组件的 props 、emits 属性上挂载对应的配置。

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

User.props = {
  userName: String
}

function User(props) {
  // const name = ref('天气好')
  return <><div>{props.userName}</div></>
}

export default User;
```

相信习惯了 React 的 fc 的小伙伴，看到这里一定感觉倍感亲切。然而 Vue 的 Jsx 终究只是 Vue 的 Jsx，它并没有像 React 一样存在那么多强大的 Hooks 和内置组件，而是仅仅只是 h 函数的便捷写法。在语法上也和 React Jsx 存在诸多区别。和 React Jsx 相比，Vue Jsx 其实和自家的 template 更接近。不过 Vue Jsx 写法的灵活性还是要比 template 模板高，但官方更推荐使用 template。template 更容易上手且提供了更好的性能优化，除非你想完完全全掌控组件的每一个细节，才需要jsx。

小结

尽管本文提到了很多种写法，但大多数写法在大多数时候都是不会派上用场的，也应该不被派上用场。之所以列举那么多写法，主要的目的还是为了循序渐进引入 jsx 文件写法和函数式组件写法。

可以看出，Vue 的写法本质上是选项式的。Vue3 在 Vue2 的基础上引入了 setup 选项和 setup 语法糖，结合组合式 api 后，开发者可以将组件的大部分逻辑都维护在 setup 中，而不是 vue2 中割裂了逻辑的 data+created+methods 选项。

在此基础上，setup 语法糖支持自动导出组件功能，为我们日常开发带来了很大的便利。

但是除了使用 template 来表达我们的结构，我们也可以自己使用 render 选项并借助 h 函数或者 jsx 的力量来手写渲染函数。这些都是在 vue 文件中完成的。

既然都不需要 template 了，那么 vue 文件里就只剩下一个 script 了（我们先忽略 style）。在 jsx 文件中，就允许我们直接书写导出对象（仍是选项式的写法），忽略script。

最后是 Vue 的 jsx 文件独有的特性。它允许我们导出一个函数作为组件，我们称之为函数式组件（fc，function component），这是 vue 文件和以前所有写法所不具备的，外形与 React 相近。

总的来说就一句话，**Vue 本身仍然是选项式的，但是它现在还额外支持了在 jsx 文件中书写 fc。**

个人看法

使用哪一种写法，主要看个人偏好。每种写法在特定场景下都有它的好处和坏处。选择哪一种并不重要。但是我还是提几点个人建议：

1. 优先选择 setup 语法糖。
2. 使用选项式时，推荐套上 defineComponent。
3. 当需要使用 jsx 时，推荐在 vue 文件中使用选项式 setup+返回 jsx 渲染函数的写法。

为什么不推荐函数式组件的写法，因为它需要写在 jsx 文件里。如果你只是想通过 jsx 为你的组件增强某些功能，直接改造 vue 文件更加方便，并且不需要修改你引入文件的后缀（`.vue` -> `.jsx/.tsx`）。最重要的是，如果真的想写函数式组件，为何不试试 React?

当然以上只是我个人的小小看法。如果觉得有用，不妨动动小手点赞和收藏吧！

看到这里，如果你也想试试在 Vue 中写 jsx，不妨看看这篇文章，分享了怎么在 Vue 项目中配置 jsx 环境，充分发挥 jsx 的优势！