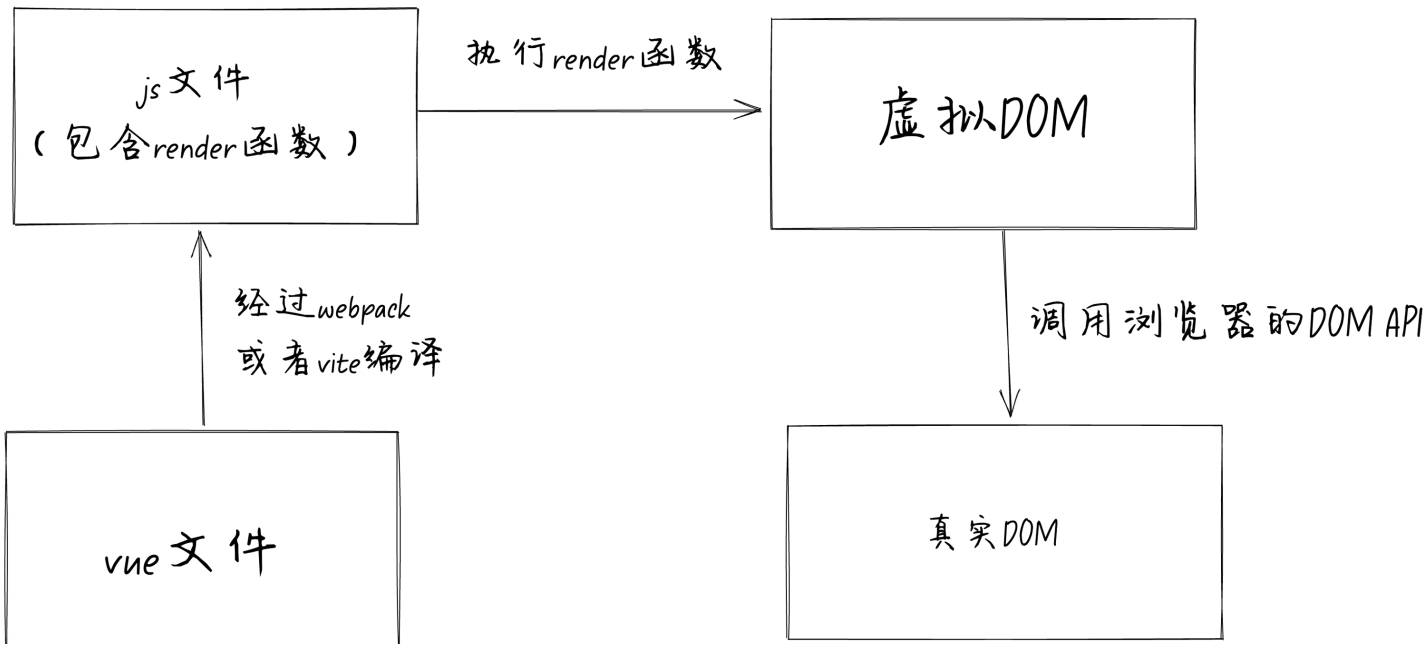
从 `vue3` 开始 `vue` 引入了宏，比如 `defineProps` 、 `defineEmits` 等。我们每天写 `vue` 代码时都会使用到这些宏，但是你有没有思考过 `vue` 中的宏到底是什么？为什么这些宏不需要手动从 `vue` 中 `import` ？为什么只能在 `setup` 顶层中使用这些宏？

vue 文件如何渲染到浏览器上

要回答上面的问题，我们先来了解一下从一个 `vue` 文件到渲染到浏览器这一过程经历了什么？

我们的 `vue` 代码一般都是写在后缀名为vue的文件上，显然浏览器是不认识vue文件的，浏览器只认识html、css、jss等文件。所以第一步就是通过 `webpack` 或者 `vite` 将一个vue文件编译为一个包含 `render` 函数的 `js` 文件。然后执行 `render` 函数生成虚拟DOM，再调用浏览器的 `DOM API` 根据虚拟DOM生成真实DOM挂载到浏览器上。



关注公众号：前端欧阳，解锁我更多 `vue` 干货文章，并且可以免费向我咨询 `vue` 相关问题。

vue3的宏是什么？

我们先来看看 `vue` 官方的解释：

> 宏是一种特殊的代码，由编译器处理并转换为其他东西。它们实际上是一种更巧妙的字符串替换形式。

宏是在哪个阶段运行？

通过前面我们知道了 `vue` 文件渲染到浏览器上主要经历了两个阶段。

第一阶段是编译时，也就是从一个 `vue` 文件经过 `webpack` 或者 `vite` 编译变成包含render函数的js文件。此时的运行环境是 `nodejs` 环境，所以这个阶段可以调用 `nodejs` 相关的 `api` ，但是没有在浏览器环境内执行，所以不能调用浏览器的 `API` 。

第二阶段是运行时，此时浏览器会执行 `js` 文件中的 `render` 函数，然后依次生成虚拟 `DOM` 和真实 `DOM` 。此时的运行环境是浏览器环境内，所以可以调用浏览器的API，但是在这一阶段中是不能调用 `nodejs` 相关的 `api` 。

而宏就是作用于编译时，也就是从vue文件编译为js文件这一过程。

举个 `defineProps` 的例子：在编译时 `defineProps` 宏就会被转换为定义 `props` 相关的代码，当在浏览器运行时自然也就没有了 `defineProps` 宏相关的代码了。所以才说宏是在编译时执行的代码，而不是运行时执行的代码。

一个 `defineProps` 宏的例子

我们来看一个实际的例子，下面这个是我们的源代码：

```
<template>
  <div>content is {{ content }}</div>
  <div>title is {{ title }}</div>
</template>

<script setup lang="ts">
import {ref} from "vue"
const props = defineProps({
  content: String,
});
const title = ref("title")
</script>
```

在这个例子中我们使用 `defineProps` 宏定义了一个类型为 `String` ，属性名为 `content` 的 `props` ，并且在 `template` 中渲染 `content` 的内容。

我们接下来再看看编译成 `js` 文件后的代码，代码我已经进行过简化：

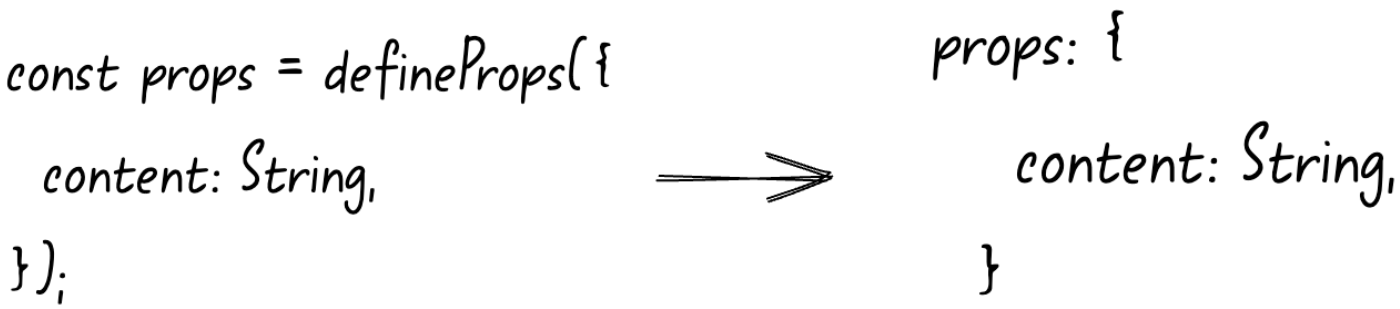
```
import { defineComponent as _defineComponent } from "vue";
import { ref } from "vue";

const __sfc__ = _defineComponent({
  props: {
    content: String,
  },
  setup(__props) {
    const props = __props;
```

```
    const title = ref("title");
    const __returned__ = { props, title };
    return __returned__;
  },
});

import {
  toDisplayString as _toDisplayString,
  createElementVNode as _createElementVNode,
  Fragment as _Fragment,
  openBlock as _openBlock,
  createElementBlock as _createElementBlock,
} from "vue";

function render(_ctx, _cache, $props, $setup) {
  return (
    _openBlock(),
    _createElementBlock(
      _Fragment,
      null,
      [
        _createElementVNode(
          "div",
          null,
          "content is " + _toDisplayString($props.content),
          1 /* TEXT */
        ),
        _createElementVNode(
          "div",
          null,
          "title is " + _toDisplayString($setup.title),
          1 /* TEXT */
        ),
      ],
      64 /* STABLE_FRAGMENT */
    )
  );
}
__sfc__.render = render;
export default __sfc__;
```

我们可以看到编译后的 `js` 文件主要由两部分组成，第一部分为执行 `defineComponent` 函数生成一个 `sfc` 对象，第二部分为一个 `render` 函数。 `render` 函数不是我们这篇文章要讲的，我们主要来看看这个 `sfc` 对象。

看到 `defineComponent` 是不是觉得很眼熟，没错这个就是 `vue` 提供的API中的 definecomponent 函数。这个函数在运行时没有任何操作，仅用于提供类型推导。这个函数接收的第一个参数就是组件选项对象，返回值就是该组件本身。所以这个 `sfc` 对象就是我们的 `vue` 文件中的 `script` 代码经过编译后生成的对象，后面再通过 `sfc.render = render` 将 `render` 函数赋值到组件对象的 `render` 方法上面。

我们这里的组件选项对象经过编译后只有两个了，分别是 `props` 属性和 `setup` 方法。明显可以发现我们原本在 `setup` 里面使用的 `defineProps` 宏相关的代码不在了，并且多了一个 `props` 属性。没错这个 `props` 属性就是我们的 `defineProps` 宏生成的。



我们再来看一个不在 `setup` 顶层调用 `defineProps` 的例子：

```
<script setup lang="ts">
import {ref} from "vue"
const title = ref("title")

if (title.value) {
  const props = defineProps({
    content: String,
  });
}
</script>
```

运行这个例子会报错：`defineProps is not defined`

我们来看看编译后的js代码：

```
import { defineComponent as _defineComponent } from "vue";
import { ref } from "vue";

const __sfc__ = _defineComponent({
  setup(__props) {
    const title = ref("title");
    if (title.value) {
      const props = defineProps({
```

```
9           content: String,
10        });
11      }
12      const __returned__ = { title };
13      return __returned__;
14    },
15  });
```

明显可以看到由于我们没有在 `setup` 的顶层调用 `defineProps` 宏，在编译时就不会将 `defineProps` 宏替换为定义 `props` 相关的代码，而是原封不动的输出回来。在运行时执行到这行代码后，由于我们没有任何地方定义了 `defineProps` 函数，所以就会报错 `defineProps is not defined` 。

总结

现在我们能够回答前面提的三个问题了。

- `vue` 中的宏到底是什么?
- `vue3` 的宏是一种特殊的代码，在编译时会将这些特殊的代码转换为浏览器能够直接运行的指定代码，根据宏的功能不同，转换后的代码也不同。
- 为什么这些宏不需要手动从 `vue` 中 `import` ?
- 因为在编译时已经将这些宏替换为指定的浏览器能够直接运行的代码，在运行时已经不存在这些宏相关的代码，自然不需要从 `vue` 中 `import` 。
- 为什么只能在 `setup` 顶层中使用这些宏?
- 因为在编译时只会去处理 `setup` 顶层的宏，其他地方的宏会原封不动的输出回来。在运行时由于我们没有在任何地方定义这些宏，当代码执行到宏的时候当然就会报错。