【前端面试】你知道Vue3中的内置组件Tele...

前言

Vue3 相较于 Vue2 而言，新增了很多新特性。其中 Teleport 内置组件就是 Vue3 新增的新特性之一，我们通常把它称为传送门，也可以把它比作哆啦 A 梦的口袋，那么这个新特性到底是干什么用的呢？感觉很神奇一般，接下来就让我们一起来学习这个新的内置组件吧！

1.基本概念

1.1 简单理解

不管是在 Vue2 还是 Vue3 中都有内置组件的存在，比如 component 内置组件、transition 内置组件等等。内置组件就是官方给我们封装的全局组件，我们直接拿来用就可以了。

在 Vue3 中新增了 Teleport 内置组件，我们先来看下官方文档是怎么解释的。

官网说明：

> <Teleport> 是一个内置组件，使我们可以将一个组件的一部分模板“传送”到该组件的 DOM 层次结构之外的 DOM 节点中。

官网说的还是很好理解了，首先就点名了这是一个内置组件，然后它的作用就是将一个组件的一些内容“传送”到另一个地方去。官网的解释可以有些词用的都比较正式，所以有些小伙伴还是不太好理解，接下来我们通俗的给大家解释一遍。

通俗解释：

> teleport 是一个内置组件，我们都知道 HTML 是由层级关系的，Vue3 中的组件也是有层级关系的。假如我们在一个父组件中引用了一个子组件，那么渲染成页面后这个子组件 HTML 也是必然被父组件 HTML 包含的。但是，如果我们把子组件放置到了 telport 组件中。那么我们就可以指定该子组件渲染到父组件之外的其它 DOM 节点下，比如 body、或者其它的 DOM 等等。这就有点类似与“传送”了。

总结下来一句话：**即使你我是父子组件关系，但是我有 teleport 包含，那么我就可以渲染到其它 DOM 节点下。**

1.2 典型案例

说再多不如看一看实际案例。我们使用 Vue 的 UI 组件库的时候，经常会用到模态框这个组件，比如我们使用 element 的模态框，我们直接在 Vite 项目里面来演示一下 Element 模态框。

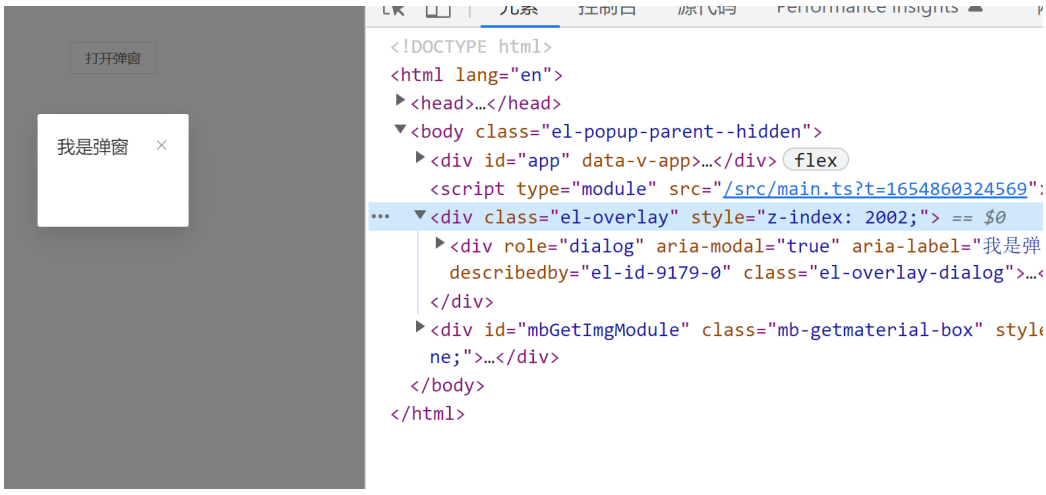
代码如下：

```
<template>
  <el-button primary @click="dialogVisible = true">打开弹窗</el-button>
  <el-dialog
    v-model="dialogVisible"
    append-to-body
    title="我是弹窗"
    width="30%"
  >
  </el-dialog>
</template>

<script lang="ts" setup>
import { ref } from "vue";
const dialogVisible = ref(false);
</script>
```

上段代码中我们在 App.vue 组件里面引用了 Element-plus 的弹窗组件，并且添加了一个 append-to-body 属性。

输出结果：



我们可以看到虽然弹窗组件是写在 App.vue 组件里面的，但是渲染出来的结果却是弹窗组件属于 body 节点，这是因为我们利用了 Element-plus 中弹窗的 append-to-body 属性，我们把该属性去掉再看看什么结果：

可以看到弹窗组件又乖乖的跑到了 App.vue 组件下面。

为何要这样做?

很简单，假如我们有非常多的弹窗，那么我们如何管理它们的 z-index 呢，也就是同时弹窗时的层级关系，如果每个弹窗都在各自的父组件中，那么我们是没法控制的，所有我们有必要把它们都拧出来，放在同一个父元素下面，这样就可以方便的设置层级关系了。

说了这么多，这和我们的 teleport 组件有什么关系吗？有很大的关系，上面弹窗的 append-to-body 属性效果是 Element 给我们做的，要是我们想自己实现这样的效果，该怎么办呢？我们就可以使用内置组件 teleport 了。

补充:

至于为什么不好控制层级关系，大家可以去了解一下 CSS层叠上下文 。

2.基础使用

了解了 teleport 的基础概念之后，我们需要学会如何使用它。我们先来学习最基本的使用方式，直接修改 App.vue 代码。

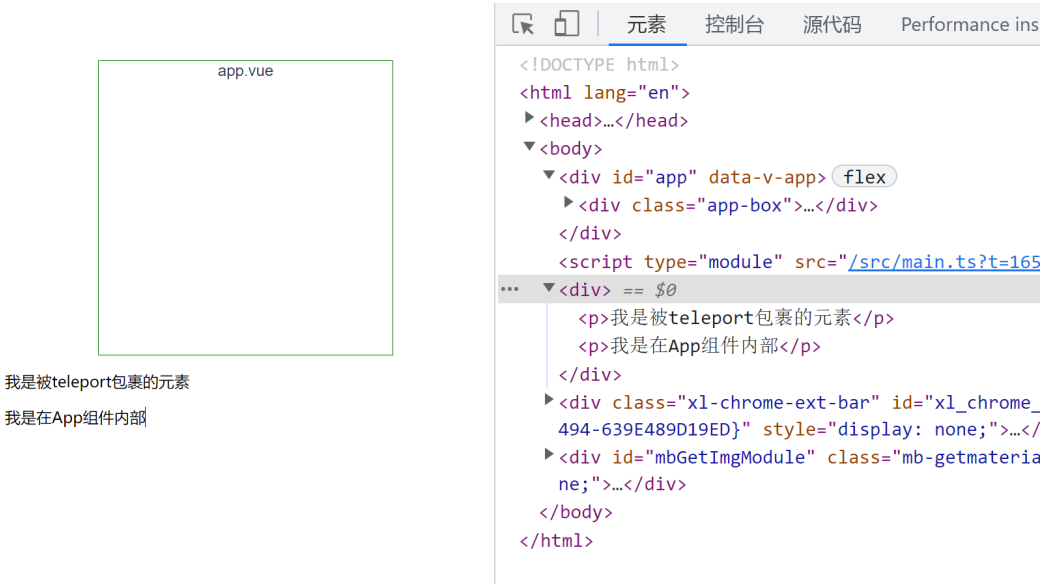
2.1 传送 DOM 节点

我们先来看看如何将 DOM 内容传送到其它地方去。

代码如下:

```
<template>
  <div class="app-box">
    app.vue
    <Teleport to="body">
      <div>
        <p>我是被 teleport 包裹的元素</p>
        <p>我是在 App 组件内部</p>
      </div>
    </Teleport>
  </div>
</template>
```

输出结果:



从上图可以看出，我们 Teleport 包裹的元素虽然是属于 app.vue 组件，但是渲染过后它却被渲染在了 body 这个 dom 元素下面了。

这都得归功于 Teleport 得传送功能，它的用法很简单，语法代码如下：

```
<Teleport to="body">
</Teleport>
```

其中 to 就是我们“传送”的目的地了，即需要把包裹的内容传送到何处去。

to 允许接收值：

> 期望接收一个 CSS 选择器字符串或者一个真实的 DOM 节点。

2.2 传送组件

Teleport 不仅仅可以传送普通的 DOM 节点，它还可以传送我们封装的组件，Element 组件中的模态框就是传送的整个组件。

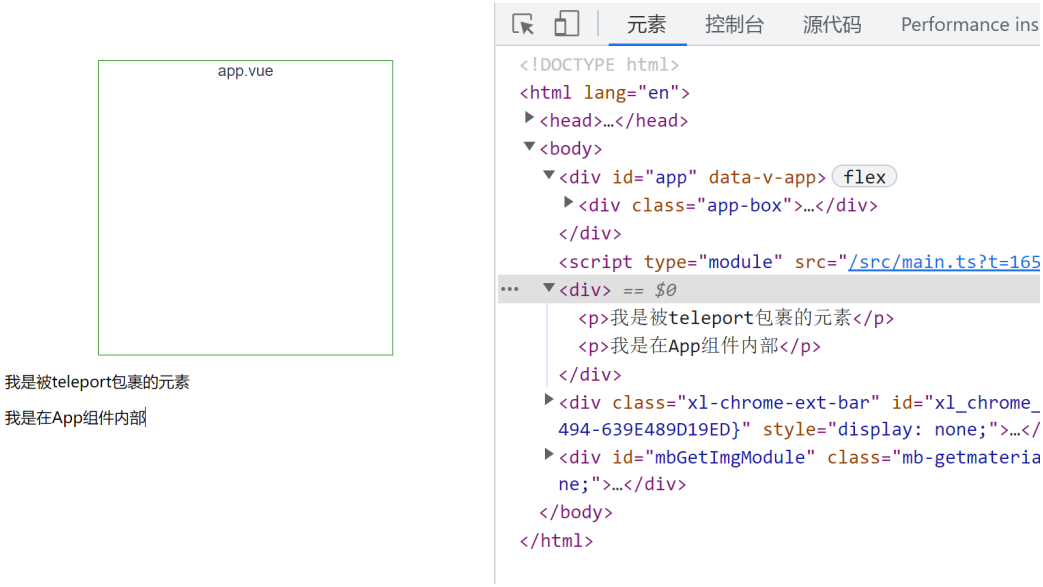
代码如下：

将刚刚的代码封装为一个 child 组件。

```
<template>
  <div class="app-box">
    app.vue
    <Teleport to="body">
      <child />
    </Teleport>
  </div>
</template>

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

输出结果：



2.3 组件间逻辑关系

很多小伙伴会有这样一个问题：既然 teleport 组件把元素传动到了其它地方，那么它和原来的父组件还有关系吗?

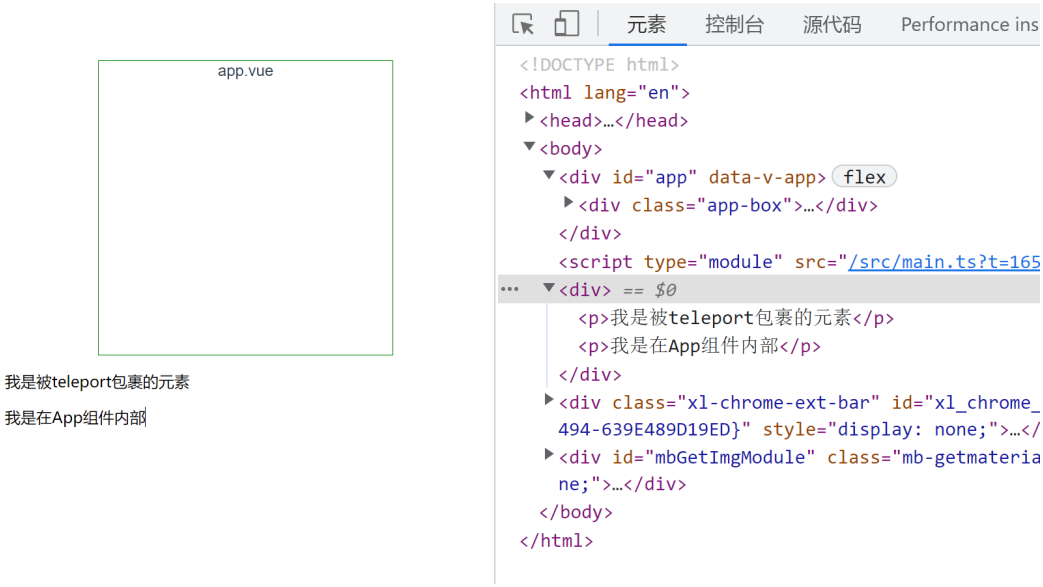
答案是肯定的!

teleport 改变的只是内部元素的渲染地方，相当于只是影响 CSS 样式，并没有更改 JS 逻辑，所以它们父子组件的逻辑还是存在的。

示例代码:

```
<template>
  <div class="app-box">
    app.vue
    <Teleport to="body">
      <div>
        <p>我是被 teleport 包裹的元素</p>
        <p>{{ message }}</p>
      </div>
    </Teleport>
  </div>
</template>
<script lang="ts" setup>
import { ref } from 'vue';
const message = ref("我是在 App 组件内部");
</script>
```

输出结果:



可以看到输出结果是一样的，说明被 teleport 包括的元素与 app.vue 组件之间的逻辑关系没有发生改变。

3.禁用传送功能

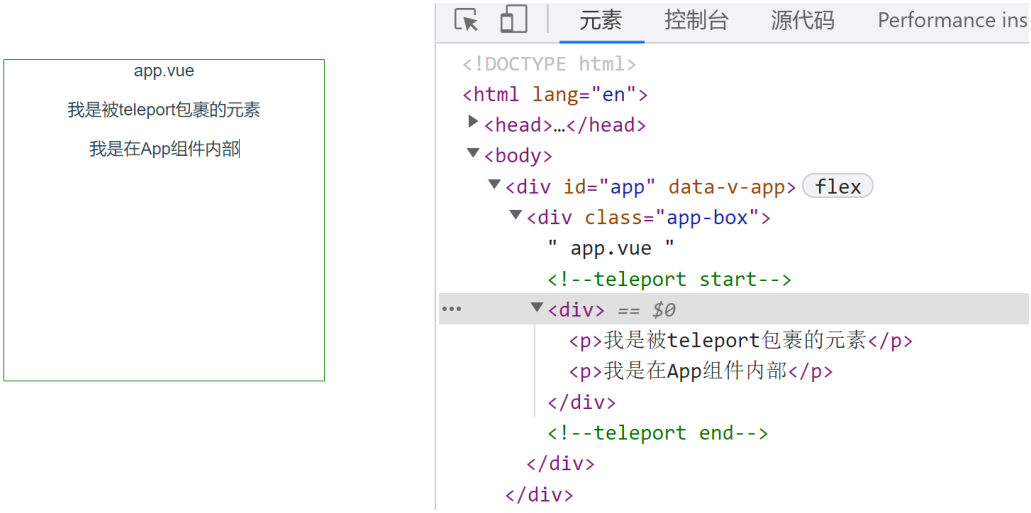
前面我们使用 Element-plus 中的弹窗组件中有一个 append-to-body 属性，它可以决定是否需要将弹窗组件传送至其它地方，这说明是否传送是可以人为控制的。

我们 teleport 内置组件也是可以控制内部元素是否需要传送的。

代码如下：

```
<template>
  <div class="app-box">
    app.vue
    <Teleport to="body" :disabled="true">
      <div>
        <p>我是被 teleport 包裹的元素</p>
        <p>{{ message }}</p>
      </div>
    </Teleport>
  </div>
</template>
<script lang="ts" setup>
import { ref } from 'vue';
const message = ref("我是在 App 组件内部");
</script>
```

输出结果：



我们给 teleport 组件添加了 disabled 属性后，就可以控制内部元素是否可以传送了，它接收一个 Boolean 值，true 代表不允许传送，false 代表传送。

这样我们就能向 element-plus 中的弹窗组件那样，人为控制是否需要开启传送了。

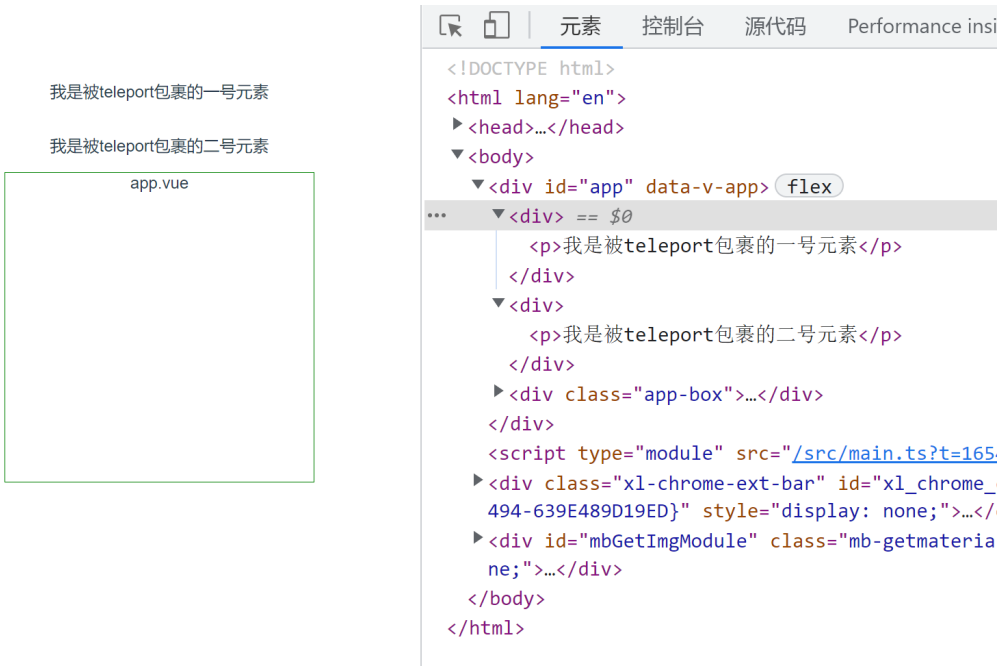
4.多个元素传送给一个节点

假如我们有很多个 teleport 组件，而且它们传送的目的地都是一个，那么它们会渲染成什么样的结果呢?

代码如下：

```
<template>
  <div class="app-box">
    app.vue
    <Teleport to="#app">
      <div>
        <p>我是被 teleport 包裹的一号元素</p>
      </div>
    </Teleport>
    <Teleport to="#app">
      <div>
        <p>我是被 teleport 包裹的二号元素</p>
      </div>
    </Teleport>
  </div>
</template>
```

输出结果：



可以看到两个元素按照顺序渲染到了 id 为 app 的元素下面。如果交换 teleport 的顺序，渲染出来的顺序也会被交换。

总结

teleport 的使用场景很多，最最经典的就是 UI 组件库中的模态框等等，比如说还有 tooltip、poptip 等等，不知道大家有没有熟悉这些组件的一个属性：transfer。当然，更多有趣的功能还需要小伙伴们自己下去挖掘，总之，teleport 这个组件不复杂，主要是它比较有趣。