【前端面试】Vue3中如何使用ref获取节点?

前言

虽然在 Vue 中不提倡我们直接操作 DOM，毕竟 Vue 的理念是以数据驱动视图。但是在实际情况中，我们有很多需求都是需要直接操作 DOM 节点的，这个时候 Vue 提供了一种方式让我们可以获取 DOM 节点：**ref 属性**。ref 属性是 Vue2 和 Vue3 中都有的，但是使用方式却不大一样，这也导致了很多从 Vue2 转到 Vue3 的小伙伴感到有些困惑。

今天我们就来揭开 Vue3 中 ref 的神秘面纱！

1.回顾 Vue2 中的 ref

在学习 Vue3 中的 ref 之前，我们先来了解下 Vue2 中 ref，这样一对比，大家更能够加深印象，以及它们之间的区别。

获取节点：

这是 ref 的基本功能之一，目的就是获取元素节点，在 Vue 中使用方式也很简单，**代码如下：**

```
<template>
  <div id="app">
    <div ref="hello">小猪课堂</div>
  </div>
</template>
<script>
export default {
  mounted() {
    console.log(this.$refs.hello); // <div>小猪课堂</div>
  },
};
</script>
```

上段代码中可以看到我们在 div 元素上绑定了 ref 属性，并命名为 hello，接下来我们直接使用 this.$refs.hello 的方式就可以获取到该 DOM 元素了。

2.Vue3 中 ref 访问元素

Vue3 中通过 ref 访问元素节点与 Vue2 不太一样，在 Vue3 中我们是没有 this 的，所以当然也没有 this.$refs。想要获取 ref，我们只能通过声明变量的方式。

创建一个 Vite 项目：

为了方便演示，我们直接在 Vite 项目中演示 ref 代码，创建项目指令如下：

```
npm create vite@latest my-vue-app --template vue-ts
```

代码如下：

```
<template>
  <div ref="hello">小猪课堂</div>
</template>
<script setup lang="ts">
import { onMounted, ref } from "vue";
const hello = ref<any>(null);
onMounted(() => {
  console.log(hello.value); // <div>小猪课堂</div>
});
</script>
```

输出结果：

```
[vite] connecting...
  <div>小猪课堂</div>
[vite] connected.
>
```

上段代码中我们同样给 div 元素添加了 ref 属性，为了获取到这个元素，我们声明了一个与 ref 属性名称相同的变量 hello，然后我们通过 hello.value 的形式便获取到了该 div 元素。

注意点：

- 变量名称必须要与 ref 命名的属性名称一致。
- 通过 hello.value 的形式获取 DOM 元素。
- 必须要在 DOM 渲染完成后才可以获取 hello.value，否则就是 null。

3.v-for 中使用 ref

使用 ref 的场景有多种，一种是单独绑定在某一个元素节点上，另一种便是绑定在 v-for 循环出来的元素上了。这是一种非常常见的需求，在 Vue2 中我们通常使用:ref="..."的形式，只要能够标识出每个 ref 不一样即可。

但是在 Vue3 中又不太一样，不过还是可以通过变量的形式接收。

代码如下：

```
<template>
  <div ref="hello">小猪课堂</div>
  <ul>
    <li v-for="item in 10" ref="itemRefs">
      {{item}} - 小猪课堂
    </li>
  </ul>
```

```
</template>
<script setup lang="ts">
import { onMounted, ref } from "vue";
const itemRefs = ref<any>([]);
onMounted(() => {
  console.log(itemRefs.value);
});
</script>
```

输出结果：

```
Proxy {0: li, 1: li, 2: li, 3: li, 4: li, 5: li, 6: li, 7: li, 8: li, 9: li}
  [[Handler]]: Object
  [[Target]]: Array(10)
    0: li
    1: li
    2: li
    3: li
    4: li
    5: li
    6: li
    7: li
    8: li
    9: li
    length: 10
    [[Prototype]]: Array(0)
  [[IsRevoked]]: false
```

上段代码中尽管是 v-for 循环，但是我们似乎使用 ref 的形式与第 2 节中的方式没有任何变化，我们同样使用变量的形式拿到了每一个 li 标签元素。

但是这里我们需要**注意一下**：我们似乎没办法区分哪个 li 标签哪个 ref，初次之外，我们的 itemRefs 数组不能够保证与原数组顺序相同，即与 list 原数组中的元素一一对应。

4.ref 绑定函数

前面我们在组件上定义 ref 时，都是以一个字符串的形式作为 ref 的名字，其实我们的 ref 属性还可以接收一个函数作为属性值，这个时候我们需要在 ref 前面加上:。

代码如下：

```
<template>
  <div :ref="setHelloRef">小猪课堂</div>
</template>
<script setup lang="ts">
import { ComponentPublicInstance, HTMLAttributes } from "vue";
const setHelloRef = (el: HTMLElement | ComponentPublicInstance | HTMLAttribu
  console.log(el); // <div>小猪课堂</div>
};
</script>
```

输出结果：

```
[vite] connecting...
▼div
  ▶__vnode: {__v_isVNode: true, __v_skip: true, type: 'div', props: {…}, key: null, …}
  ▶__vueParentComponent: {uid: 0, vnode: {…}, type: {…}, parent: null, appContext: {…}, …}
   accessKey: ""
   align: ""
   ariaAtomic: null
   ariaAutoComplete: null
   ariaBusy: null
   ariaChecked: null
   ariaColCount: null
   ariaColIndex: null
   ariaColSpan: null
   ariaCurrent: null
```

上段代码中 ref 属性接收的是一个 setHelloRef 函数，该函数会默认接收一个 el 参数，这个参数就是我们需要获取的 div 元素。假如需求中我们采用这种方式的话，那么完全可以把 el 保存到一个变量中去，供后面使用。

那么，我们在 v-for 中是否也能采用这种方式呢?

答案是可以的!

v-for 中使用

代码如下:

```
<template>
  <ul>
    <li v-for="item in 10" :ref="(el) => setItemRefs(el, item)">
      {{ item }} - 小猪课堂
    </li>
  </ul>
</template>
<script setup lang="ts">
import { ComponentPublicInstance, HTMLAttributes, onMounted } from "vue";
let itemRefs: Array<any> = [];
const setItemRefs = (el: HTMLElement | ComponentPublicInstance | HTMLAttribu
  if(el) {
    itemRefs.push({
      id: item,
      el,
    });
  }
}
onMounted(() => {
  console.log(itemRefs);
});
</script>
```

输出结果:

```
[vite] connecting...
▼(10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
  ▶0: {id: 1, el: li}
  ▶1: {id: 2, el: li}
  ▶2: {id: 3, el: li}
  ▶3: {id: 4, el: li}
  ▶4: {id: 5, el: li}
  ▶5: {id: 6, el: li}
  ▶6: {id: 7, el: li}
  ▶7: {id: 8, el: li}
  ▶8: {id: 9, el: li}
  ▶9: {id: 10, el: li}
    length: 10
  ▶[[Prototype]]: Array(0)
[vite] connected.
```

在 v-for 中使用函数的形式传入 ref 与不使用 v-for 时的形式差不多，不过这里我们做了一点变通，为了区别出哪个 ref 是哪一个 li 标签，我们决定将 item 传入函数，也就是(el) => setItemRefs(el, item)的写法。

这种形式的好处既让我们的操作性变得更大，还解决了 v-for 循环是 ref 数组与原数组顺序不对应的问题。

5.组件上使用 ref

前面我们所使用 ref 时，都是在一个具体的 dom 元素上绑定，但是我们也可以将 ref 绑定在组件上，比如在 Vue2 中，我们将 ref 绑定在组件上时，便可以获取到该组件里面的所有数据和方法.

虽然 Vue3 中也可以将 ref 绑定在组件上，但是具体能获取组件的哪些值还是有一些区别的，我们一起来看看。

代码如下：

```
<template>
  <child ref="childRef"></child>
</template>
<script setup lang="ts">
import { onMounted, ref } from "vue";
import child from "./child.vue";
const childRef = ref<any>(null);
onMounted(() => {
  console.log(childRef.value); // child 组件实例
  console.log(childRef.value.message); // undefined
});
</script>
```

子组件 child 代码：

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

```
</template>
<script lang="ts" setup>
import { ref } from "vue";
const message = ref<string>("我是子组件");
const onChange = () => {};
</script>
```

输出结果：

```
[vite] connecting...
▶ Proxy {__v_skip: true}
undefined
[vite] connected.
>
```

上段代码中我们新增了一个子组件，然后再子组件上面绑定了 ref，其用法基本上和 ref 直接绑定在 DOM 元素上一致。

但是如果我们把 ref 绑定再组件上，通常就是为了调用子组件里面的方法或者数据，可是从上面的输出结果来看，我们没有获取到数据，即 childRef.value.message 为 undefined，这也是与 Vue2 的不同之处。

在 Vue3 中，使用 ref 获取子组件时，如果想要获取子组件的数据或者方法，子组件可以通过 defineExpose 方法暴露数据。

修改子组件代码：

```
<template>
  <div>{{ message }}</div>
</template>
<script lang="ts" setup>
import { ref } from "vue";
const message = ref<string>("我是子组件");
const onChange = () => {
  console.log("我是子组件方法")
};
defineExpose({
  message,
  onChange
});
</script>
```

父组件再次获取：

```
const childRef = ref<any>(null);
onMounted(() => {
  console.log(childRef.value); // child 组件实例
  console.log(childRef.value.message); // 我是子组件
  childRef.value.onChange(); // 我是子组件方法
```

```
});
```

输出结果：

```
[vite] connecting...
▶ Proxy {message: RefImpl, __v_skip: true, onChange: f}
我是子组件
我是子组件方法
[vite] connected.
>
```

可以看到我们在父组件中可以获取到子组件暴露的数据和方法了。

总结

虽然 Vue2 和 Vue3 中的 ref 使用方式有着较大的区别，但是它们的目的都是一样的，所以我们只要朝着目的前进，都会与美好相遇的！