Template Refs

While Vue's declarative rendering model abstracts away most of the direct DOM operations for you, there may still be cases where we need direct access to the underlying DOM elements. To achieve this, we can use the special `ref` attribute:

```template
<input ref="input">
```

`ref` is a special attribute, similar to the `key` attribute discussed in the `v-for` chapter. It allows us to obtain a direct reference to a specific DOM element or child component instance after it's mounted. This may be useful when you want to, for example, programmatically focus an input on component mount, or initialize a 3rd party library on an element.

Accessing the Refs

To obtain the reference with Composition API, we can use the `useTemplateRef()` 3.5+ helper:

```vue
<script setup>
import { useTemplateRef, onMounted } from 'vue'

// the first argument must match the ref value in the template
const input = useTemplateRef('my-input')

onMounted(() => {
  input.value.focus()
})
</script>

<template>
  <input ref="my-input" />
</template>
```

When using TypeScript, Vue's IDE support and `vue-tsc` will automatically infer the type of `input.value` based on what element or component the matching `ref` attribute is used on.

Note that you can only access the ref **after the component is mounted.** If you try to access `input` in a template expression, it will be `null` on the first render. This is because the element doesn't exist until after the first render!

If you are trying to watch the changes of a template ref, make sure to account for the case where the ref has `null` value:

```js
watchEffect(() => {
  if (input.value) {
    input.value.focus()
  } else {
    // not mounted yet, or the element was unmounted (e.g. by v-if)
  }
})
```

See also: Typing Template Refs TS

Ref on Component

> This section assumes knowledge of Components. Feel free to skip it and come back later.

`ref` can also be used on a child component. In this case the reference will be that of a component instance:

```vue
<script setup>
import { useTemplateRef, onMounted } from 'vue'
import Child from './Child.vue'

const childRef = useTemplateRef('child')

onMounted(() => {
  // childRef.value will hold an instance of <Child />
})
</script>

<template>
  <Child ref="child" />
</template>
```

▶ Usage before 3.5

If the child component is using Options API or not using `<script setup>`, the referenced instance will be identical to the child component's `this`, which means the parent component will have full access to every property and method of the child component. This

should try to implement parent / child interactions using the standard props and emit interfaces first.

An exception here is that components using `<script setup>` are **private by default**: a parent component referencing a child component using `<script setup>` won't be able to access anything unless the child component chooses to expose a public interface using the `defineExpose` macro:

```vue
<script setup>
import { ref } from 'vue'

const a = 1
const b = ref(2)

// Compiler macros, such as defineExpose, don't need to be imported
defineExpose({
  a,
  b
})
</script>
```

When a parent gets an instance of this component via template refs, the retrieved instance will be of the shape `{ a: number, b: number }` (refs are automatically unwrapped just like on normal instances).

Note that defineExpose must be called before any await operation. Otherwise, properties and methods exposed after the await operation will not be accessible.

See also: Typing Component Template Refs TS

Refs inside `v-for`

> Requires v3.5 or above

When `ref` is used inside `v-for`, the corresponding ref should contain an Array value, which will be populated with the elements after mount:

```vue
<script setup>
import { ref, useTemplateRef, onMounted } from 'vue'

const list = ref([
  /* ... */
])

const itemRefs = useTemplateRef('items')
```

```
</script>

<template>
  <ul>
    <li v-for="item in list" ref="items">
      {{ item }}
    </li>
  </ul>
</template>
```

Try it in the Playground

▶ Usage before 3.5

It should be noted that the ref array does **not** guarantee the same order as the source array.

Function Refs

Instead of a string key, the `ref` attribute can also be bound to a function, which will be called on each component update and gives you full flexibility on where to store the element reference. The function receives the element reference as the first argument:

```template
<input :ref="(el) => { /* assign el to a property or ref */ }">
```

Note we are using a dynamic `:ref` binding so we can pass it a function instead of a ref name string. When the element is unmounted, the argument will be `null`. You can, of course, use a method instead of an inline function.

Edit this page on GitHub

Previous
Watchers

Next
Components Basics