侦听器

需要在状态变化时执行一些“副作用”：例如更改 DOM，或是根据异步操作的结果去修改另一处的状态。使用 `watch` 函数在每次响应式状态发生变化时触发回调函数：

```
// 监听用户离开后再回到本页面的情况
watch(pageVisibility, (value) => {
  if (value === "visible") {
  }
});
```

```
watch(
  () => route.path,
  () => {
    // console.log('route.path', route.path);
    if (route?.path?.includes("/exam/examInfo")) {
    }
  },
  { immediate: true, deep: true }
);
```

`watch` 默认是懒执行的：仅当数据源变化时，才会执行回调。但在某些场景中，我们希望在创建侦听器时，立即执行一遍回调。举例来说，我们想请求一些初始数据，然后在相关状态更改时重新请求数据。

我们可以通过传入 `immediate: true` 选项来强制侦听器的回调立即执行：

```
watch(
  source,
  (newValue, oldValue) => {
    // 立即执行，且当 `source` 改变时再次执行
  },
  { immediate: true }
);
```

watchEffect()

侦听器的回调使用与源完全相同的响应式状态是很常见的。例如下面的代码，在每当 `todoId` 的引用发生变化时使用侦听器来加载一个远程资源：

```
const todoId = ref(1);
const data = ref(null);

watch(
```

```
  todoId,
  async () => {
    const response = await fetch(
      `https://jsonplaceholder.typicode.com/todos/${todoId.value}`
    );
    data.value = await response.json();
  },
  { immediate: true }
);
```

特别是注意侦听器是如何两次使用 todoId 的，一次是作为源，另一次是在回调中。

```
watchEffect(async () => {
  const response = await fetch(
    `https://jsonplaceholder.typicode.com/todos/${todoId.value}`
  );
  data.value = await response.json();
});
```

回调会立即执行，不需要指定 `immediate: true`。在执行期间，它会自动追踪 `todoId.value` 作为依赖（和计算属性类似）。每当 `todoId.value` 变化时，回调会再次执行。有了 `watchEffect()`，我们不再需要明确传递 `todoId` 作为源值。

`watch` 和 `watchEffect`

`watch` 和 `watchEffect` 都能响应式地执行有副作用的回调。它们之间的主要区别是追踪响应式依赖的方式：

`watch` 只追踪明确侦听的数据源。它不会追踪任何在回调中访问到的东西。另外，仅在数据源确实改变时才会触发回调。`watch` 会避免在发生副作用时追踪依赖，因此，我们能更加精确地控制回调函数的触发时机。

`watchEffect`，则会在副作用发生期间追踪依赖。它会在同步执行过程中，自动追踪所有能访问到的响应式属性。这更方便，而且代码往往更简洁，但有时其响应性依赖关系会不那么明确。

停止侦听器

在 `setup()` 或 `<script setup>` 中用同步语句创建的侦听器，会自动绑定到宿主组件实例上，并且会在宿主组件卸载时自动停止。

如果用异步回调创建一个侦听器，那么它不会绑定到当前组件上，你必须手动停止它，以防内存泄漏。

```
<script setup>
import { watchEffect } from 'vue';
// 它会自动停止
watchEffect(() => {})

// ...这个则不会!
setTimeout(() => {
  watchEffect(() => {})
```

```
}, 100)
</script>
```

要手动停止一个侦听器，请调用 `watch` 或 `watchEffect` 返回的函数：

```
const unwatch = watchEffect(() => {});
// ...当该侦听器不再需要时
unwatch();
```