【前端面试】Vue3中的watch和watchEffec...

前言

使用过Vue的小伙伴，不管时Vue2还是Vue3，我相信你都用过Vue中的监听器。监听器的作用就和它的名字一样：用来监听某个东西是否发生变化！我们很多需求都会用到监听器watch，但是Vue2和Vue3中的监听器的用法有些许不一样，这就让一些从Vue2转Vue3的小伙伴不太适应，所以，我们今天就来好好学一学Vue3中的监听器如何使用！

1.环境准备

为了方便演示和编写代码，我们直接使用vite搭建一个Vue3的基础项目。

创建命令：

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

删除App.vue中一些不需要的东西，然后运行项目：

2.watch

2.1 watch基本使用

在Vue3中的组合式API中，watch的作用和Vue2中的watch作用是一样的，他们都是用来监听响应式状态发生变化的，当响应式状态发生变化时，都会触发一个回调函数。

代码如下：

```
<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <p>{{ message }}</p>
  <button @click="changeMsg">更改message</button>
</template>
<script setup lang="ts">
import { ref, watch } from "vue";
const message = ref("小猪课堂");
watch(message, (newValue, oldValue) => {
  console.log("新的值:", newValue);
  console.log("旧的值:", oldValue);
});
const changeMsg = () => {
  message.value = "张三";
};
</script>
```

上段代码中我们点击按钮就会更改响应式变量message的值。我们又使用watch监听器监听了message变量，当它发生变化时，就会触发watch监听函数中的回调函数，并且回调函数默认接收两个参数：新值和旧值。

注意：当我们第一进入页面时，watch监听函数的回调函数是不会执行的。

输出结果：

2.2 watch监听类型

前面我们一直强调watch监听的是响应式数据，如果我们监听的数据不是响应式的，那么可能会抛出如下警告：

```
▶[Vue warn]: Invalid watch source:  小猪课堂 A watch source can only be a getter/effect function, a ref, a reactive object, or an array of these types.
  at <App>
```

那么哪些数据是属于响应式的，或者换个说法，watch监听器可以监听哪些形式的数据呢?

（1）ref和计算属性

ref定义的数据我们是可以监听到的，因为我们前面的代码以及证明了。除此之外，计算属性也是可以监听到的，比如下列代码：

```
const message = ref("小猪课堂");
const newMessage = computed(() => {
  return message.value;
});
watch(newMessage, (newValue, oldValue) => {
  console.log("新的值:", newValue);
  console.log("旧的值:", oldValue);
});
```

当我们message发生变化时，计算属性newMessage也会重新计算得出新的结果，我们watch监听函数是可以监听到计算属性变化的。

（2）getter函数

这里的getter函数大家可以简单的理解为获取数据的一个函数，说白了该函数就是一个返回值的操作，有点类似与计算属性。

示例代码如下：

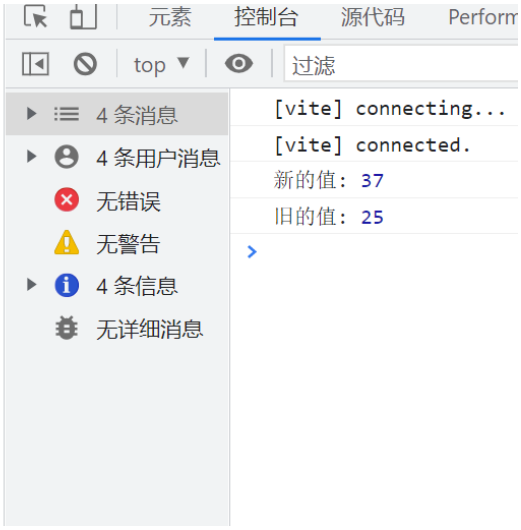
```
<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <p>{{ message }}</p>
  <p>{{ x1 + x2 }}</p>
  <button @click="changeMsg">更改message</button>
</template>
<script setup lang="ts">
import { ref, watch } from "vue";
const message = ref("小猪课堂");
const x1 = ref(12);
const x2 = ref(13);
watch(
  () => x1.value + x2.value,
  (newValue, oldValue) => {
    console.log("新的值:", newValue);
    console.log("旧的值:", oldValue);
  }
);
const changeMsg = () => {
  message.value = "张三";
  x1.value = 14;
  x2.value = 23;
};
</script>
```

输出结果：

张三

37

更改message



上段代码中watch监听器中的第一个参数是一个箭头函数，也就是getter函数，getter函数返回的是响应式数据x1和x2相加的值，当这两个中中有一个变化，都会执行watch中的回调函数。有点像是直接把计算属性写到监听器里面去了。

（3）监听响应式对象

前面我们监听的都是值类型的响应式数据，我们同样也可以监听响应式的对象。

代码如下：

```
const number = reactive({ count: 0 });
const countAdd = () => {
  number.count++;
};
watch(number, (newValue, oldValue) => {
  console.log("新的值:", newValue);
  console.log("旧的值:", oldValue);
});
```

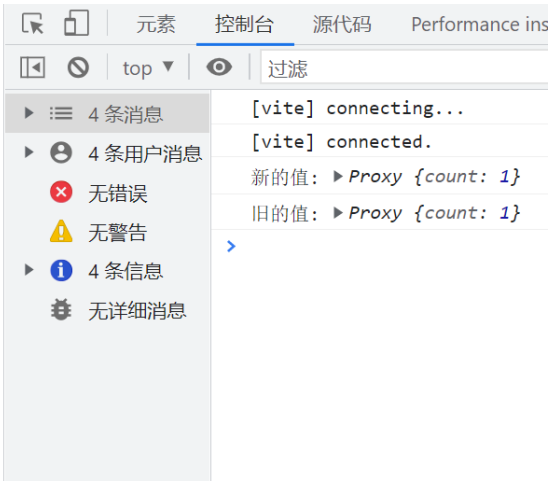
输出结果：

小猪课堂

25

更改message

增加count



当watch监听的是一个响应式对象时，会隐式地创建一个深层侦听器，即该响应式对象里面的任何属性发生变化，都会触发监听函数中的回调函数。

需要注意的，watch不能直接监听响应式对象的属性，即下面的写法是错误的：

```
const number = reactive({ count: 0 });
```

```
const countAdd = () => {
  number.count++;
};
watch(number.count, (newValue, oldValue) => {
  console.log("新的值:", newValue);
  console.log("旧的值:", oldValue);
});
```

上段代码中相当于你直接向watch传递了一个非响应式的数字，然而watch只能监听响应式数据。

但是：

如果我们非要监听响应式对象中的某个属性，我们可以使用getter函数的形式，代码如下：

```
watch(
  () => number.count,
  (newValue, oldValue) => {
    console.log("新的值:", newValue);
    console.log("旧的值:", oldValue);
  }
);
```

上段代码也是可以监听到count变化的。

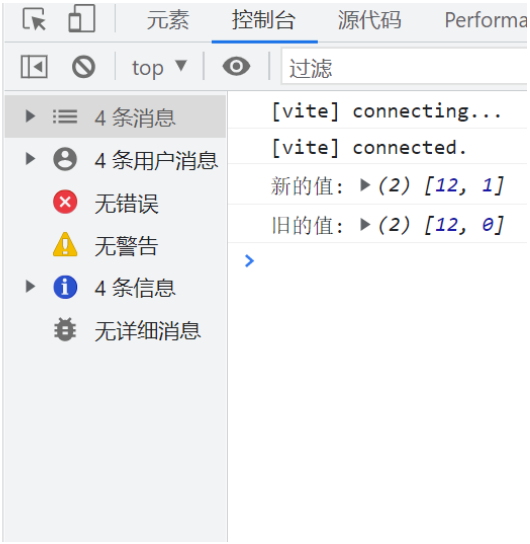
（4）监听多个来源的数组

watch还可以监听数组，前提是这个数组内部含有响应式数据。

代码如下：

```
const x1 = ref(12);
const number = reactive({ count: 0 });
const countAdd = () => {
  number.count++;
};
watch([x1, () => number.count], (newValue, oldValue) => {
  console.log("新的值:", newValue);
  console.log("旧的值:", oldValue);
});
```

输出结果：



2.3 深度监听

在前面的代码中，如果我们将一个响应式对象传递给watch监听器时，只要对象里面的某个属性发生了变化，那么就会执行监听器回调函数。

究其原因，因为我们传入响应对象给watch时，隐式的添加一个深度监听器，这就让我们造成了我们牵一发而至全身的效果。

但是，如果我们是使用的getter函数返回响应式对象的形式，那么响应式对象的属性值发生变化，是不会触发watch的回调函数的。

代码如下：

```
const number = reactive({ count: 0 });
const countAdd = () => {
  number.count++;
};
watch(
  () => number,
  (newValue, oldValue) => {
    console.log("新的值:", newValue);
    console.log("旧的值:", oldValue);
  },
);
```

上段代码中我们使用getter函数返回了响应式对象，当我们更改number中count的值时，watch的回调函数是不会执行的。

为了实现上述代码的监听，我们可以手动给监听器加上深度监听的效果。

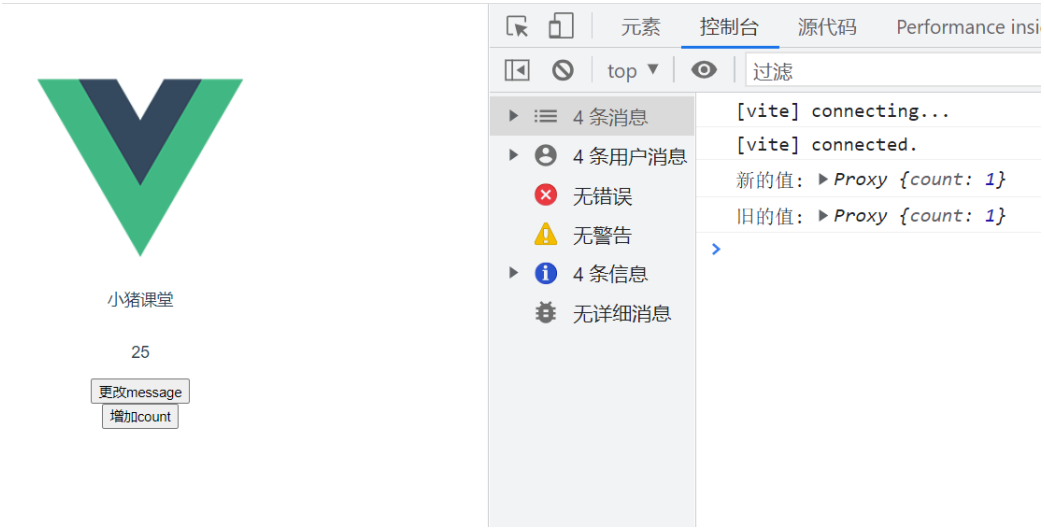
代码如下：

```
const number = reactive({ count: 0 });
const countAdd = () => {
  number.count++;
};
watch(
```

```
  () => number,
  (newValue, oldValue) => {
    console.log("新的值:", newValue);
    console.log("旧的值:", oldValue);
  },
  { deep: true }
);
```

添加深度监听很简单，只需要给watch添加第三个参数即可：{ deep: true }。

输出结果：



注意：上段代码中的newValue和oldValue的值是一样的，除非我们把响应式对象即number整个替换掉，那么这两个值才会变得不一样。除此之外，深度监听会遍历响应式对象的所有属性，开销较大，当对象体很大时，需要慎用。

所以我们推荐getter函数只返回相应是对象中的某一个属性！！

3.watchEffect

我们前面使用watch监听数据状态时，不知道大家有没有发现这样一个问题：只有当我们监听的数据源发生了变化，监听函数的回调函数才会执行。但是需求总是多变的，有些场景下我们可能需要刚进页面，或者说第一次渲染页面的时候，watch监听器里面的回调函数就执行一遍。

面对这种需求我们怎样处理呢？一般有两种方式：

方式一：

这种方式也是通过watch实现的，确切的说是巧妙的实现，而不是依赖于watch监听器。

代码如下：

```
const number = reactive({ count: 0 });
// 进入页面先执行一遍
const callback = () => {
  console.log("新的值:", number.count);
  console.log("旧的值:", number.count);
};
callback();
```

```
watch(
  () => number.count,
  (newValue, oldValue) => {
    console.log("新的值:", newValue);
    console.log("旧的值:", oldValue);
  },
  { deep: true }
);
```

既然我们想要第一次进入页面的时候就执行一遍回调函数，那么我们不妨把回调函数直接提取出来，进入页面执行一遍即可，这也算是巧妙的实现了我们的需求。

但是这种方式似乎不太优雅，而且有些繁琐。所以Vue推出了更加优雅的方法：watchEffect监听器。

方式二：

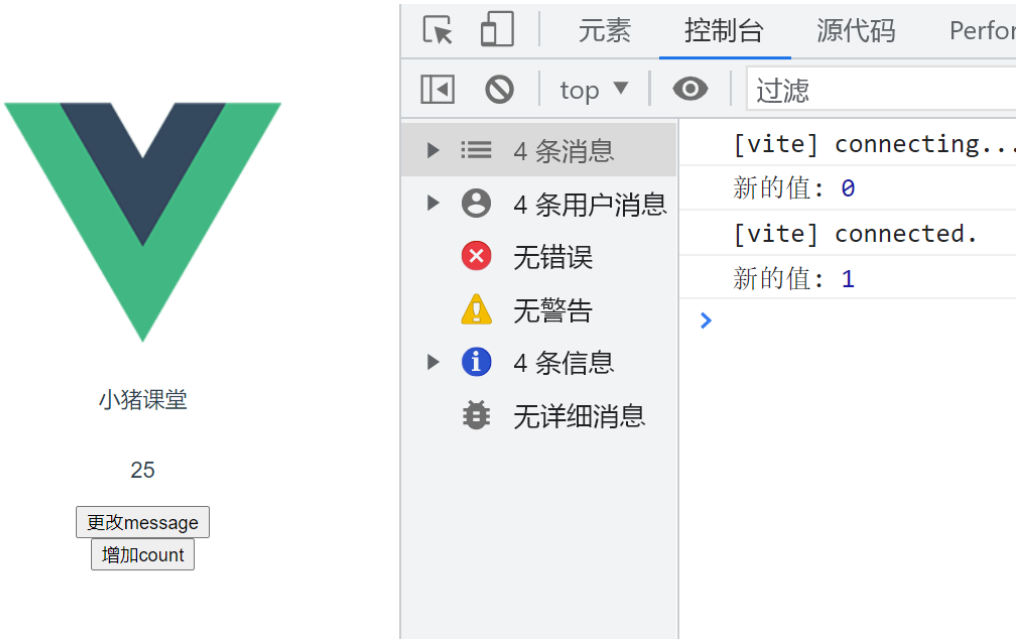
watchEffect也是一个监听器，只不过它不会像watch那样接收一个明确的数据源，它只接收一个回调函数。而在这个回调函数当中，它会自动监听响应数据，当回调函数里面的响应数据发生变化，回调函数就会立即执行。

所以我们可以将方式一中的代码使用watchEffect优雅的实现。

代码如下：

```
const number = reactive({ count: 0 });
const countAdd = () => {
  number.count++;
};
watchEffect(()=>{
  console.log("新的值:", number.count);
})
```

输出结果：



上段代码中，当我们第一次进入页面时，number响应数据从无到有，这个时候就会触发watchEffect的回调函数，因为在watchEffect回调函数中使用了number响应数据，所以它会自动跟踪number数据的变化。当我们点击按钮更改count的值时，watchEffect中的回调函数便会再次执行。

这样代码是不是简单很多呀！

4.watch和watchEffect区别

我们已经大概知道了watch和watchEffect的用法，那么它们之间的区别相信大家也了解了一些，这里我们总结一下它们之间的区别。

- watch和watchEffect都能监听响应式数据的变化，不同的是它们监听数据变化的方式不同。
- watch会明确监听某一个响应数据，而watchEffect则是隐式的监听回调函数中响应数据。
- watch在响应数据初始化时是不会执行回调函数的，watchEffect在响应数据初始化时就会立即执行回调函数。

5.回调中的DOM

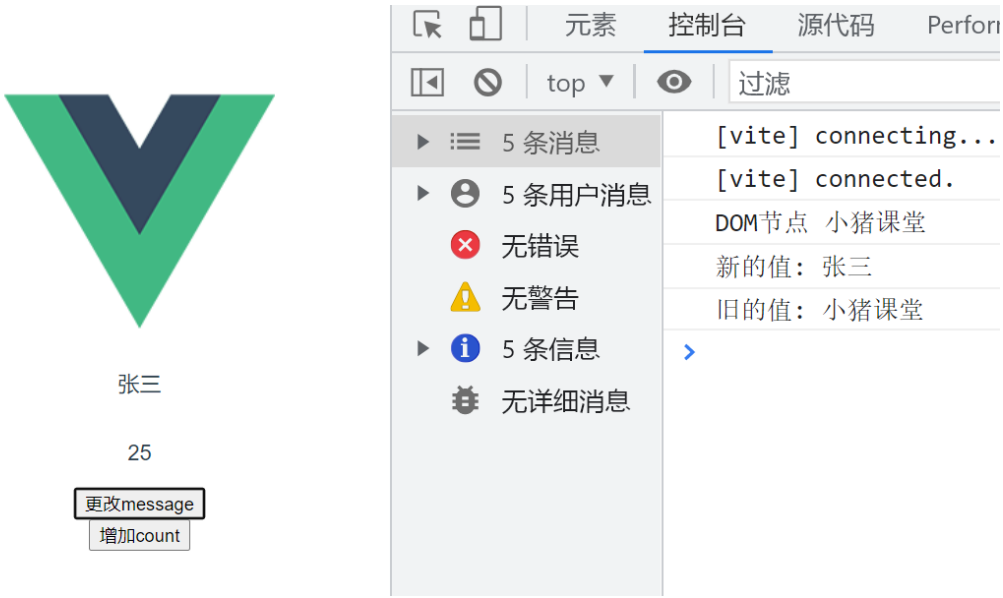
在前面的讲解中，我们忽视了一个问题：如果我们在监听器的回调函数中或取DOM，这个时候的DOM是更新前的还是更新后的?

我们不妨实验一下。

代码如下：

```
<template>
  <img alt="Vue logo" src="./assets/logo.png" />
  <p ref="msgRef">{{ message }}</p>
  <button @click="changeMsg">更改message</button>
</template>
<script setup lang="ts">
import { computed, reactive, ref, watch, watchEffect } from "vue";
const message = ref("小猪课堂");
const msgRef = ref<any>(null);
const changeMsg = () => {
  message.value = "张三";
};
watch(message, (newValue, oldValue) => {
  console.log("DOM节点", msgRef.value.innerHTML);
  console.log("新的值:", newValue);
  console.log("旧的值:", oldValue);
});
</script>
```

输出结果：



我们通过点击按钮更改message的值，从“小猪课堂”变为“张三”。但是我们发现在监听器的回调函数里面获取到的DOM元素还是“小猪课堂”，说明DOM还没有更新。

解决方法：

如果我们想要在回调函数里面获取更新后的DOM，非常简单，我们只需要再给监听器多传递一个参数选项即可：flush: ’post’。watch和watchEffect同理。

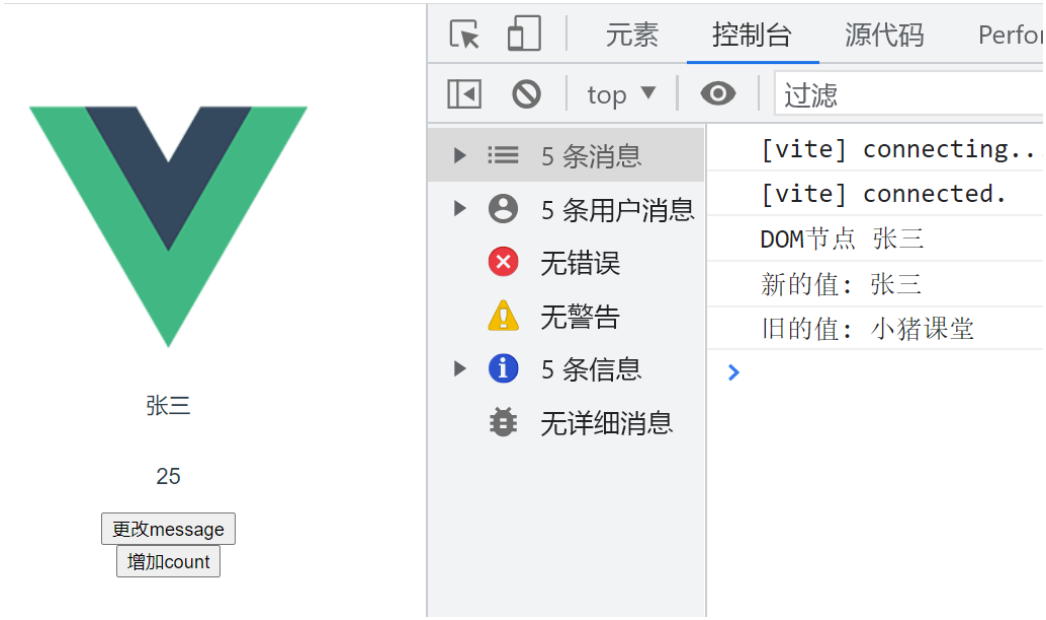
代码如下：

```
watch(source, callback, {
  flush: 'post'
})
watchEffect(callback, {
  flush: 'post'
})
```

修改后的代码：

```
watch(
  message,
  (newValue, oldValue) => {
    console.log("DOM节点", msgRef.value.innerHTML);
    console.log("新的值:", newValue);
    console.log("旧的值:", oldValue);
  },
  {
    flush: "post",
  }
);
```

输出结果：



这个时候我们在回调函数中获取到的已经是更新后的DOM节点了。

补充：

虽然watch和watchEffect都可以用上述方法解决DOM问题，但是Vue3单独给watchEffect提供了一个更方便的方法，也可以叫做watchEffect的别名，代码如下：

```
watchPostEffect(() => {
  /* 在 Vue 更新后执行 */
})
```

6.手动停止监听器

通常来说，我们的一个组件被销毁或者卸载后，监听器也会跟着被停止，并不需要我们手动去关闭监听器。但是总是有一些特殊情况，即使组件卸载了，但是监听器依然存在，这个时候其实式需要我们手动关闭它的，否则容易造成内存泄漏。

比如下面这中写法，我们就需要手动停止监听器：

```
<script setup>
import { watchEffect } from 'vue'
// 它会自动停止
watchEffect(() => {})
// ...这个则不会！
setTimeout(() => {
  watchEffect(() => {})
}, 100)
</script>
```

上段代码中我们采用异步的方式创建了一个监听器，这个时候监听器没有与当前组件绑定，所以即使组件销毁了，监听器依然存在。

关闭方法很简单，**代码如下：**

```
const unwatch = watchEffect(() => {})
```

```
// ...当该侦听器不再需要时
unwatch()
```

我们需要用一个变量接收监听器函数的返回值，其实就是返回的一个函数，然后我们调用该函数，即可关闭当前监听器。

总结

有些小伙伴第一次接触Vue3中的监听器时，感觉有点难，摸不着头脑！其实无非就是两个函数，作用稍微不一样而已，重点我们还是需要理解组合式API的思想，这样学起来就简单多了。