Vue 生命周期钩子完整指南

mp.weixin.qq.com/s/KrtUc_5Itd7ymTxmUQfdgw

喜欢就关注我们吧

> 本文翻译自：
> https://learnvue.co/2020/12/how-to-use-lifecycle-hooks-in-vue3/

Vue 2 和 Vue 3 中的生命周期钩子，其工作方式非常相似。

如果我们在项目中使用Options API，那就不必更改Vue生命周期钩子的任何代码，因为Vue 3兼容Vue之前的版本。

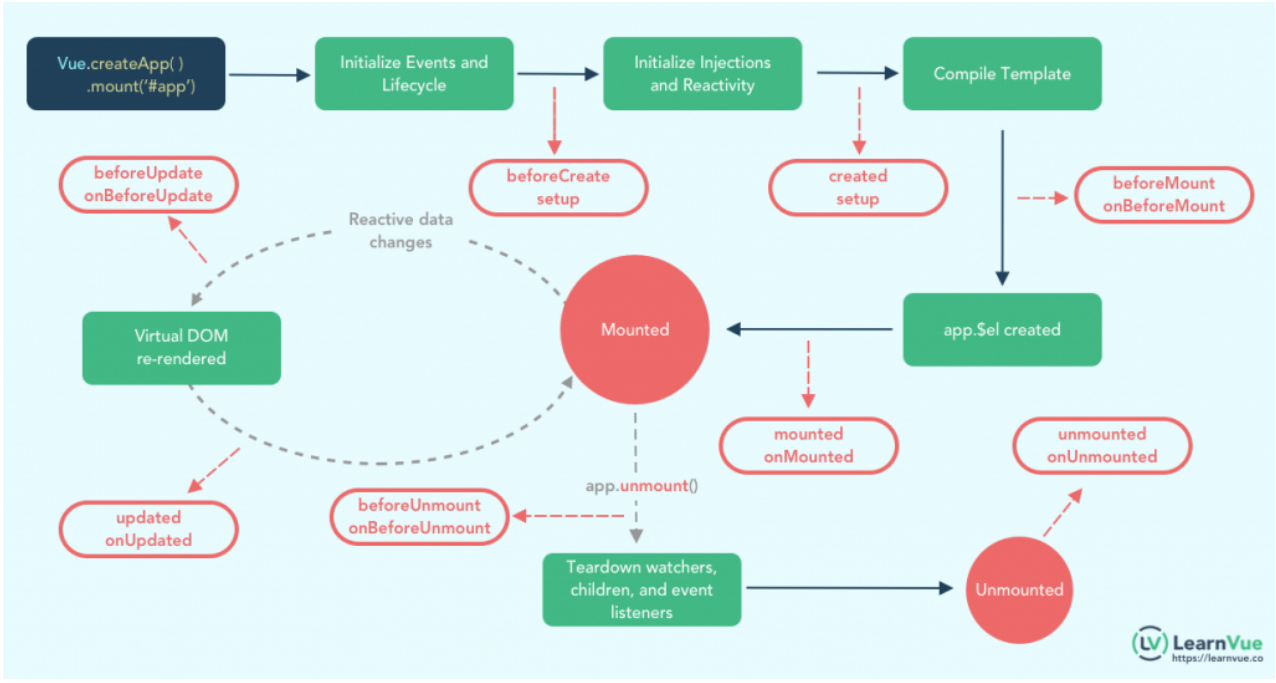
不过，如果使用的是Composition API，那么访问这些钩子的方式略有不同。

这篇文章旨在帮助大家了解如何在Options API和Composition API中使用生命周期钩子，从而写出更优的代码。

一起来看看吧！

什么是Vue生命周期钩子

首先，让我们来看一下关于Options API和Composition API中Vue 3生命周期钩子的图文概述。



从本质上说，每个主要的Vue生命周期事件都被分成两个钩子，在事件之前和事件之后被调用。你可以在Vue app中应用四个主要事件（也就是有8个主要钩子）。

- `Creation`：当组件被创建时运行
- `Mounting`：在挂载DOM时运行
- `Updates`：在修改响应数据时运行
- `Destruction`：在销毁元素前运行。

在Options API中使用Vue生命周期钩子

使用Options API时，生命周期钩子作为选项提供给Vue实例。我们不需要导入任何东西，只需调用该方法并为该生命周期钩子编写代码即可。

例如，假设我们要访问`mount()`和`updated()`生命周期钩子。代码如下：

```
<script>
   export default {
      mounted() {
         console.log('mounted!')
      },
      updated() {
         console.log('updated!')
      }
   }
</script>
```

够简单了吧？

OK，现在让我们到Composition API中使用Vue 3生命周期钩子。

在Vue 3 Composition API中使用Vue生命周期钩子

在Composition API中，我们必须先将生命周期钩子导入到项目中，然后才能使用它们。这是为了尽可能保持项目的轻量级。

```
import { onMounted } from 'vue'
```

除了 `beforeCreate` 和 `created` （它们被 `setup` 方法本身所替代），我们可以在 `setup` 方法中访问9个Options API生命周期钩子。

- `onBeforeMount` ：在挂载开始前调用
- `onMounted` ：在挂载组件后调用
- `onBeforeUpdate` ：在响应性数据更新和重新渲染之前调用
- `onUpdated` ：重新渲染后调用
- `onBeforeUnmount` ：在销毁Vue实例之前调用
- `onUnmounted` ：在实例销毁后调用
- `onActivated` ：在激活保持活动的组件时调用
- `onDeactivated` ：当停用保持活动的组件时调用
- `onErrorCaptured` ：从子组件捕获错误时调用

当我们导入这些钩子并在代码中访问时，如下所示：

```
<script>
import { onMounted } from'vue'

 export default {
   setup () {
     onMounted(() => {
       console.log('mounted in the composition api!')
     })
   }
}
</script>
```

更新Vue 2代码为Vue 3生命周期钩子

这个便捷的Vue 2到Vue 3生命周期映射就来自于Vue 3 Composition API文档，非常有用。

- `beforeCreate` -> 使用 `setup()`
- `created` -> 使用 `setup()`
- `beforeMount` -> `onBeforeMount`
- `mounted` -> `onMounted`
- `beforeUpdate` -> `onBeforeUpdate`
- `updated` -> `onUpdated`
- `beforeDestroy` -> `onBeforeUnmount`
- `destroyed` -> `onUnmounted`
- `errorCaptured` -> `onErrorCaptured`

深入了解生命周期钩子

我们现在知道了两个重点：

- 我们可以使用不同生命周期钩子
- 如何在Options API和Composition API中使用生命周期钩子

接下来让我们深入了解每个生命周期钩子，看看是它们如何使用的，我们可以在每个钩子中编写什么样的代码，以及它们在Options API和Composition API中的区别。

组件创建钩子——VueJS生命周期的开始

组件创建钩子是在程序运行要做的第一件事。

`beforeCreate()` ——Options API

由于组件创建钩子能初始化所有响应性数据和事件，所以 `beforeCreate` 不能访问任何组件的响应性数据和事件。

以下面的代码块为例：

```
export default {
   data() {
     return {
       val: 'hello'
     }
   },
   beforeCreate() {
     console.log('Value of val is: ' + this.val)
   }
}
```

`val` 的输出值是 `undefined` ，因为数据尚未初始化。你也不能在这个方法中调用组件方法。

如果你想看可用内容的完整列表，建议只运行 `console.log(this)` 来查看已初始化的内容。当使用Options API时，这在所有其他钩子中也有用。

当你需要某种不需要分配给数据的逻辑/API调用时，使用 `beforeCreate` 钩子很有用。因为如果我们现在给数据分配一些东西，一旦状态被初始化数据就会丢失。

`created()` ——Options API

我们现在可以访问组件的数据和事件。因此，修改上面的示例以便使用 `created` 代替 `beforeCreate` ，并查看输出如何变化。

```
export default {
   data() {
     return {
       val: 'hello'
     }
   },
   created() {
     console.log('Value of val is: ' + this.val)
   }
}
```

输出为：`Value of val is: hello`，因为已经初始化了数据。

在处理读取/写入响应性数据时，使用`created`方法很有用。例如，如果你想进行API调用，然后存储该值，就到了`created`方法大显身手的时候了。

此处，`created`比`mounted`更好，是因为这是在Vue同步初始化过程的早期，并且你可以随意执行数据读/写。

至于Composition API Creation钩子怎么样？

对于使用Composition API的Vue 3生命周期钩子，`beforeCreate`和`created`都被`setup()`方法替换。这意味着你本应放入这些方法中的任何代码现在都位于`setup`方法中。

我们刚刚在创建生命周期钩子中编写的代码得重写：

```
import { ref } from'vue'

 export default {
   setup() {
     const val = ref('hello')
     console.log('Value of val is: ' + val.value)
     return {
       val
     }
   }
}
```

挂载钩子——访问DOM

这些挂载钩子处理挂载和渲染组件，是项目和应用程序中最常用的一些钩子。

`beforeMount()`和`onBeforeMount()`

在实际渲染和挂载组件DOM之前调用。在这一步中，根元素还不存在。在Options API中，可以通过`this.$el`访问。在Composition API中，你必须在根元素上使用`ref`才能执行此操作。

```
export default {
   beforeMount() {
     console.log(this.$el)
   }
}
```

使用 `ref` 的Composition模板如下所示。

```
<template>
   <div ref='root'>
     Hello World
   </div>
</template>
```

然后，对应的脚本尝试访问 `ref` 。

```
import { ref, onBeforeMount } from'vue'

 export default {
   setup() {
      const root = ref(null)
      onBeforeMount(() => {
         console.log(root.value)
      })
      return {
         root
      }
    },
    beforeMount() {
      console.log(this.$el)
    }
 }
```

由于尚未创建app.$el，因此输出将为 `undefined` 。

`mount()` 和 `onMounted()`

在第一次渲染组件后调用。此时元素可用于直接的DOM访问。

在Options API中，我们可以使用 `this.$el` 来访问DOM，而在Composition API中，我们需要使用 `refs` 来访问Vue生命周期钩子中的DOM。

```
import { ref, onMounted } from'vue'


exportdefault {
   setup() {    /* Composition API */


const root = ref(null)


     onMounted(() => {
console.log(root.value)
     })

              return {
       root
     }
   },
   mounted() { /* Options API */
     console.log(this.$el)
   }
 }
```

更新钩子——VueJS生命周期中的响应性

每当修改响应性数据时都会触发 `updated` 生命周期事件，从而触发渲染更新。

`beforeUpdate()` 和 `onBeforeUpdate()`

在数据更改时，且在重新渲染组件之前运行。这是在发生任何更改之前手动更新DOM的好地方。例如，你可以删除事件侦听器。

`beforeUpdate` 可用于跟踪对组件所做的编辑次数，甚至跟踪创建undo功能的操作。

`update()` 和 `onUpdate()`

一旦DOM更新，就会调用 `updated` 方法。下面是一些同时使用 `beforeUpdate` 和 `updated` 的入门代码。

```
<template>
    <div>
      <p>{{val}} | edited {{ count }} times</p>
      <button @click='val = Math.random(0, 100)'>Click to Change</button>
    </div>
 </template>
```

使用相应的脚本：

```
export default {
  data() {
     return {
       val: 0
     }
  },
  beforeUpdate() {
     console.log("beforeUpdate() val: " + this.val)
  },
  updated() {
     console.log("updated() val: " + this.val
  }
}
```

或者

```
import { ref, onBeforeUpdate, onUpdated } from'vue'


exportdefault {
   setup () {
const count = ref(0)
const val = ref(0)


     onBeforeUpdate(() => {
       count.value++;
console.log("beforeUpdate");
     })


     onUpdated(() => {
console.log("updated() val: " + val.value)
     })

             return {
       count, val
     }
   }
}
```

这些方法很有用，但是我们可能要使用`watcher`来检测数据变化以应对很多用例。`watcher`很不错，因为可以给出被更新数据的旧值和新值。

另一种选择是使用计算值来更改基于元素的状态。

销毁钩子——清理

组件的销毁钩子用于移除所有组件的过程。如果处理不当，删除事件侦听器和清理可能导致内存泄漏。

`beforeUnmount()` 和 `onBeforeUnmounted()`

这在组件开始销毁之前调用，此时进行大部分（如果不是全部）清理工作。在这个阶段，组件仍然是完全有功效的，也没有破坏任何东西。

在Options API中删除事件侦听器的示例如下。

```
export default {
   mounted() {
     console.log('mount')
     window.addEventListener('resize', this.someMethod);
   },
   beforeUnmount() {
     console.log('unmount')
     window.removeEventListener('resize', this.someMethod);
   },
   methods: {
      someMethod() {
         // do smth
      }
   }
}
```

下面是在Composition API中的实现

```
import { onMounted, onBeforeUnmount } from'vue'



exportdefault {
   setup () {



const someMethod = () => {
// do smth
     }



     onMounted(() => {
console.log('mount')
window.addEventListener('resize', someMethod);
     })



     onBeforeUnmount(() => {
console.log('unmount')
window.removeEventListener('resize', someMethod);
     })

                  }
 }
```

你可以通过在Vite、vue-cli或任何支持热重载的开发环境中工作来查看。当代码更新时，某些组件将自行卸载和挂载。

`unmounted()` 和 `onUnmounted()`

此时，大部分组件及其属性都已消失。所以现在我们来打印一些数据来查看到底还剩什么，以及对项目是否有用。

```
import { onUnmounted } from'vue'


exportdefault {
  setup () { /* Composition API */


    onUnmounted(() => {
console.log('unmounted')
    })

        },
  unmounted() { /* Options API */
    console.log('unmounted')
  }
}
```

激活钩子——管理保持活动的组件

保持活动标签是动态组件的包装元素。它存储对非活动组件的缓存引用，以便Vue不必在每次动态组件改变时都要创建一个全新的实例。

对于这个特定的用例，Vue为我们提供了两个生命周期钩子

`activate()` 和 `onActivated()`

每当保持活动的动态组件重新激活时都会调用此方法——这意味着它现在是动态组件的活动视图。

例如，如果我们使用保持活动的组件来管理不同的选项卡视图，每次我们在选项卡之间切换时，当前选项卡都会运行这个激活钩子。

假设我们使用保持活动的包装器设置了以下动态组件。

```
<template>
<div>
<span @click='tabName = "Tab1"'>Tab 1 </span>
<span @click='tabName = "Tab2"'>Tab 2</span>
<keep-alive>
<component :is='tabName' class='tab-area'/>
</keep-alive>
</div>
</template>

                                  <script>


import Tab1 from'./Tab1.vue'
import Tab2 from'./Tab2.vue'



import { ref } from'vue'



exportdefault {
components: {
    Tab1,
    Tab2
  },
  setup () { /* Composition API */
const tabName = ref('Tab1')

           return {
      tabName
    }
  }
}
</script>
```

在`Tab1.vue`组件中，我们可以像这样访问激活钩子。

```
<template>
<div>
<h2>Tab 1</h2>
<input type='text' placeholder='this content will persist!'/>
</div>
</template>

    <script>


import { onActivated } from'vue'

 export default {
 setup() {
    onActivated(() => {
       console.log('Tab 1 Activated')
    })
 }
}
</script>
```

`deactivated()` 和 `onDeactivated()`

顾名思义，这在保持活动的组件不再是动态组件的活动视图时调用。

此钩子可用于在特定视图失去焦点保存用户数据和触发动画等用例。

我们可以像这样捕获钩子。

```
import { onActivated, onDeactivated } from'vue'



exportdefault {
  setup() {
    onActivated(() => {
console.log('Tab 1 Activated')
    })

            onDeactivated(() => {
       console.log('Tab 1 Deactivated')
    })
  }
}
```

现在，当我们在选项卡之间切换时——将会缓存和保存每个动态组件的状态。

Vue 3调试钩子

Vue 3为我们提供了两个可用于调试的钩子，分别是：

- `onRenderTracked`
- `onRenderTriggered`

这两个事件都采取了 `DebuggerEvent` ，允许我们判断是什么导致了Vue实例中的重新渲染。

```
export default {
    onRenderTriggered(e) {
       debugger
       // inspect which dependency is causing the component to re-render
    }
}
```

总结

无论你决定使用Options API还是Composition API，重要的是不仅要知道要使用什么生命周期钩子，还要知道为什么要使用它。

生命周期钩子可以帮助解决很多问题，但关键是要选择最适合当前用例的。

希望这能帮助大家更彻底地了解生命周期钩子以及如何在项目中的实现它。

祝编码快乐！

每日分享前端插件干货，欢迎关注！

前端新世界

分享 JS / CSS 技术教程；Vue、React、jQuery等前端开发组件

543篇原创内容

公众号

点赞和在看就是最大的支持❤️