## Módulo 1: Introdução ao Vue.js

- Vuejs  lifecycle hooks, o ciclo de vida

O **ciclo de vida** em Vue.js refere-se às fases pelas quais cada instância de componente Vue passa durante sua existência. Cada componente em Vue tem uma série de etapas desde o momento em que é criado, montado no DOM, atualizado e finalmente destruído.

O Vue.js fornece **lifecycle hooks** (ganchos de ciclo de vida) que permitem que os desenvolvedores executem código em momentos específicos do ciclo de vida do componente. Esses hooks são extremamente úteis para tarefas como buscar dados de uma API ao montar um componente ou liberar recursos ao destruir um componente.

### Fases do Ciclo de Vida:

1. **Criação (Creation):**
    - A instância Vue é inicializada e os dados observáveis são definidos, mas o componente ainda não foi montado no DOM.
2. **Montagem (Mounting):**
    - O componente é montado no DOM, ou seja, seus templates e dados são renderizados na página.
3. **Atualização (Updating):**
    - Quando os dados observáveis do componente mudam, o componente é re-renderizado.
4. **Destruição (Destruction):**
    - O componente é removido do DOM e todos os ouvintes de eventos, reações e bindings são limpos.

Agora vamos explorar os principais hooks com exemplos práticos.

### Principais Lifecycle Hooks:

1. `beforeCreate`: Chamado assim que a instância do componente é inicializada, antes de qualquer configuração de dados ou métodos.
2. `created`: Chamado após a criação da instância e a configuração dos dados observáveis, mas o componente ainda não está montado no DOM.
3. `beforeMount`: Chamado antes de o DOM virtual ser montado.
4. `mounted`: Chamado quando o DOM real foi montado e o componente está visível na página.
5. `beforeUpdate`: Chamado antes de a DOM ser re-renderizada devido a mudanças nos dados.
6. `updated`: Chamado após a DOM ser re-renderizada.
7. `beforeDestroy`: Chamado antes de a instância do componente ser destruída.

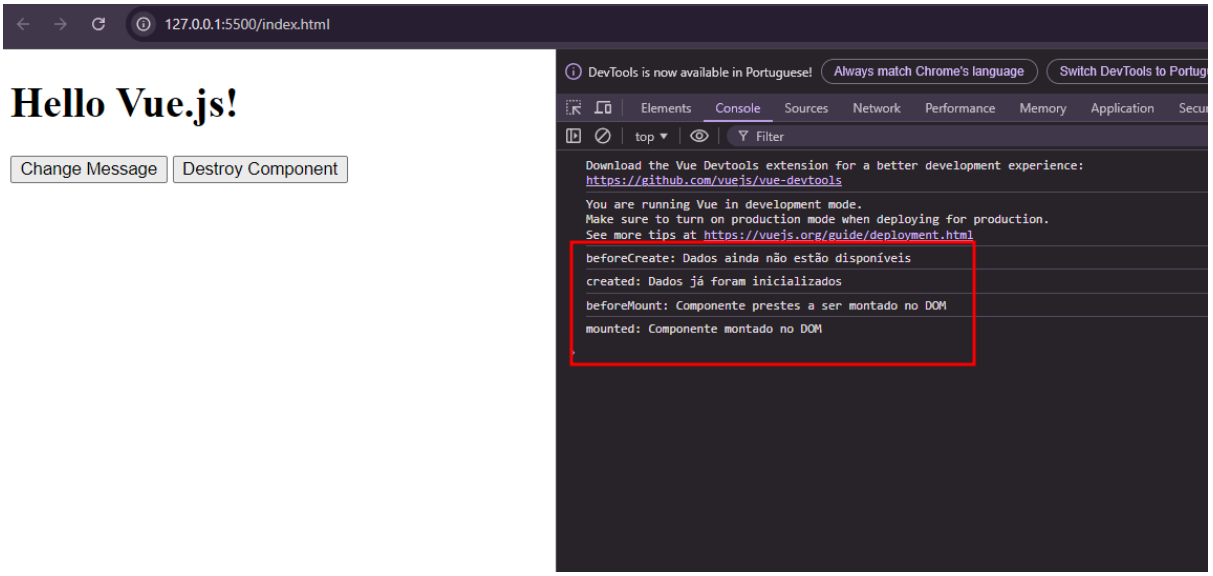
8. `destroyed`: Chamado após a destruição do componente e a remoção do DOM.

### Exemplo Prático

Aqui está um exemplo simples que demonstra o uso de alguns dos hooks do ciclo de vida:

```
    beforeUpdate() {
      console.log('beforeUpdate: Dados foram alterados, prestes a atualizar o DOM');
    },
    updated() {
      console.log('updated: DOM foi atualizado');
    },
    beforeDestroy() {
      console.log('beforeDestroy: Componente prestes a ser destruído');
    },
    destroyed() {
      console.log('destroyed: Componente foi destruído');
    }
  });
</script>

</body>
</html>
```




**Explicação:**

1. **beforeCreate**: Nesse estágio, os dados e métodos ainda não estão disponíveis. Você pode ver isso no console.
2. **created**: Agora os dados (`message`) estão disponíveis. Este é o local ideal para fazer requisições API, pois os dados estão prontos, mas o componente ainda não foi montado.
3. **beforeMount**: Chamado antes de o DOM ser montado.
4. **mounted**: O componente está totalmente montado no DOM. Esse é o lugar ideal para interagir com o DOM diretamente, caso seja necessário.
5. **beforeUpdate**: Quando o botão "Change Message" é clicado, o valor de `message` muda, o que aciona o hook `beforeUpdate`.
6. **updated**: Depois que a mudança de `message` é refletida no DOM, o hook `updated` é chamado.

7. **beforeDestroy**: Quando o botão "Destroy Component" é clicado, o Vue aciona o `beforeDestroy` e logo em seguida o `destroyed`, removendo o componente do DOM.
8. **destroyed**: O componente foi removido do DOM e todas as instâncias e ouvintes foram destruídos.

### Uso Prático de Hooks:

1. `created`: Ideal para chamadas a API ou inicialização de dados.

```
created() {
  this.fetchData();
},
methods: {
  fetchData() {
    // Simulando uma chamada API
    setTimeout(() => {
      this.message = 'Dados carregados da API';
    }, 2000);
  }
}
```

`mounted`: Útil para interagir com o DOM ou iniciar bibliotecas baseadas no DOM (por exemplo, inicializar gráficos, sliders).

```
mounted() {
  console.log('Componente montado, DOM pronto');
}
```

`beforeDestroy`: Para liberar recursos, como listeners de eventos ou limpar timers.

```
beforeDestroy() {
  console.log('Limpando listeners ou timers');
}
```

**Conclusão:**

O ciclo de vida do Vue.js oferece controle em diversos momentos do desenvolvimento de um componente. Você pode executar código durante a criação, montagem, atualização ou destruição do componente, o que permite uma enorme flexibilidade para manipular dados, fazer chamadas API, interagir com o DOM e muito mais.

O uso dos hooks de ciclo de vida depende da necessidade do seu componente, mas entender como cada um funciona ajuda a escrever código mais eficiente e limpo.