Escreva um código que aprimore todos os arrays de forma que você possa chamar o método array.last() em qualquer array e ele retornará o último elemento. Se não houver elementos na matriz, ela deve retornar -1.

Você pode assumir que a matriz é a saída de JSON.parse.

Resolução

Essa com certeza é a forma mais curta de resolver o problema:

```
return this.length ? this.slice(-1)[0] : -1;
```

Porém, há outras duas maneiras de pegar o último elemento do Array:

1°: `this.length - 1`

2°: `this.pop()`

Todas elas são válidas e podem funcionar para obter o último elemento de um array. A escolha entre elas pode depender da legibilidade, desempenho e de como você prioriza os princípios de código limpo. Aqui está uma avaliação de cada opção em termos de código limpo:

`this.length - 1`

Esta abordagem utiliza diretamente o índice do último elemento. Embora seja concisa e relativamente clara, pode não ser imediatamente óbvio para alguém lendo o código o que -1 significa neste contexto. Para manutenção, é uma boa prática usar métodos ou variáveis mais descritivos.

`this.pop()`

Usar o `pop()` para recuperar e remover o último elemento pode funcionar, mas tenha em mente que `pop()` na verdade modifica o array, o que pode não ser o que você deseja. Se você só deseja recuperar o último elemento sem alterar o array original, essa pode não ser a melhor escolha.

`this.slice(-1)[0]`

Esta abordagem usa o método `.slice()` para criar um novo array apenas com o último elemento e, em seguida, recupera esse elemento do novo array usando `[0]`. É um pouco mais longa, mas é muito clara em sua intenção. Ela não modifica o array original e indica claramente que você está interessado no último elemento.

Em termos de código limpo e legibilidade, a terceira opção `this.slice(-1)[0]` é geralmente considerada melhor porque comunica claramente sua intenção sem modificar o array original. No entanto, se você estiver trabalhando com arrays grandes e o desempenho for uma preocupação, a primeira opção `[this.length - 1]` pode ser ligeiramente mais eficiente.

Em resumo, a melhor solução depende do contexto específico do seu código, do nível de clareza que você deseja alcançar e de quaisquer considerações de desempenho. Sempre busque um código que seja fácil de entender, manter e modificar, e considere os compromissos entre diferentes opções.