Challenge 2704 -> To Be or Not To Be

From LeetCode

Escreva uma função expect que ajude os desenvolvedores a testar seu código. Ele deve receber qualquer valor val e retornar um objeto com as duas funções a seguir.

toBe(val) aceita outro valor e retorna true se os dois valorizam === um ao outro. Se eles não forem iguais, deve gerar um erro "Not Equal".

notToBe(val) aceita outro valor e retorna true se os dois valorizam !== um ao outro. Se forem iguais, deve gerar um erro "Equal".

Explicação

1. A função expect é definida com um parâmetro val, que pode ser qualquer valor.
2. A função expect retorna um objeto que contém dois métodos: toBe e notToBe.
3. O método toBe compara o valor val com um valor expected passado como argumento. Se eles forem iguais, retorna true, caso contrário, lança um erro indicando "Not Equal".
4. O método notToBe compara o valor val com um valor expected passado como argumento. Se eles forem diferentes, retorna true, caso contrário, lança um erro indicando "Equal".
5. Vários testes são executados para demonstrar o funcionamento dos métodos toBe e notToBe.
6. Cada teste é colocado em um bloco try...catch para capturar possíveis erros lançados pelas funções toBe e notToBe.

Essencialmente, meu código implementa um sistema simples de verificação de expectativas, onde você pode usar os métodos toBe e notToBe para testar igualdade e desigualdade entre o valor fornecido à função expect e um valor esperado. Se a comparação falhar, um erro é lançado, e eu posso capturá-lo para lidar com a situação de forma controlada.

Observações

Se você remover o bloco try e catch, o código ainda funcionará, mas a diferença é que, se uma exceção for lançada, ela não será capturada e tratada pelo código, o que pode resultar em uma interrupção abrupta da execução do programa.

O uso do bloco try...catch é altamente recomendado quando você espera que uma operação possa lançar exceções e deseja lidar com essas exceções de forma controlada, em vez de deixar o programa parar de funcionar completamente.

Sem o try...catch, se uma exceção for lançada, ela se propagará pela pilha de chamadas de função até encontrar um ponto onde está sendo capturada por um bloco try...catch, ou até chegar à parte superior da pilha, onde o programa geralmente irá quebrar e parar de executar.

Portanto, o uso do try...catch não é estritamente necessário, mas é uma prática recomendada para lidar com exceções de maneira mais elegante e controlada, permitindo que você tome medidas apropriadas em caso de erro, como exibir mensagens de erro amigáveis ou continuar a execução de outras partes do código mesmo que uma parte esteja com problemas.

Se você chamar expect(5).notToBe(5) fora de um bloco try...catch, isso resultará em uma exceção sendo lançada, o que pode interromper a execução normal do seu programa. A presença ou ausência de um bloco try...catch ainda é relevante nesse caso.

Se você quer que seu código continue funcionando mesmo se exceções forem lançadas durante os testes, é uma boa prática usar o bloco try...catch para capturar essas exceções e lidar com elas de maneira controlada. Caso contrário, o lançamento de exceções pode levar a problemas de funcionalidade e depuração mais difíceis.

Por que um Objeto?

Usar um objeto para retornar o resultado da função expect é uma abordagem comum quando se trata de criar um sistema de testes ou expectativas personalizadas em JavaScript. Existem algumas razões para isso:

Encadeamento de Métodos: Ao retornar um objeto com métodos como toBe, você pode encadear várias expectativas em uma única linha de código. Isso permite uma sintaxe mais legível e expressiva ao escrever testes. Por exemplo:

expect(5).toBe(5).toEqual(5).toBeGreaterThan(0);

Namespace e Organização: Usar um objeto permite criar um namespace para suas funções de teste. Isso evita conflitos de nome e ajuda a organizar suas expectativas de teste sob um único objeto. Em um cenário de teste mais complexo, você pode ter várias funções de expectativa, e agrupá-las em um objeto torna a estrutura mais organizada.

Flexibilidade: Retornar um objeto oferece mais flexibilidade para adicionar mais métodos de expectativa no futuro. Por exemplo, você pode adicionar métodos como toThrow, toBeInstanceOf, etc., para estender suas opções de teste.

Readabilidade: Usar um objeto com nomes de métodos descritivos pode tornar o código mais legível e semântico. A intenção do teste é mais clara quando você vê expect(5).toBe(5) em comparação com uma sintaxe mais obscura.

No meu exemplo original, o objeto retornado pela função expect contém um método toBe, mas, como mencionei anteriormente, isso pode ser expandido para incluir outros métodos de expectativa à medida que suas necessidades de teste evoluem.