Introdução a objetos em JavaScript

Samuel Amaro

1 de maio de 2021

Herança Em JavaScript

Com a maior parte dos detalhes principais do OOJS agora explicados, este artigo mostra como criar classes de objetos "child"(construtores) que herdam recursos de suas classes "parent". Além disso, apresentamos alguns conselhos sobre quando e onde você pode usar o OOJS e veja como as classes são tratadas na sintaxe moderna do ECMAScript.

Objetivo: Entender como é possível implementar a herança em Javascript.

Herança Prototipada

Até agora vimos alguma herança em ação — vimos como funcionam as cadeias de protótipos e como os membros são herdados subindo em uma cadeia. Mas principalmente isso envolveu funções internas do navegador. Como criamos um objeto em JavaScript que herda de outro objeto?

Vamos explorar como fazer isso com um exemplo concreto.

Quando você usaria a herança em JavaScript ?

Particularmente após este último artigo, você pode estar pensando "woo, isso é complicado". Bem, você está certo. Protótipos e herança representam alguns dos aspectos mais complexos do JavaScript, mas muito do poder e flexibilidade do JavaScript vem de sua estrutura e herança de objetos, e vale a pena entender como ele funciona.

De certa forma, você usa herança o tempo todo. Sempre que você usa vários recursos de uma API da Web ou métodos / propriedades definidos em um objeto de navegador interno que você chama em suas cadeias de caracteres, matrizes, etc., você está implicitamente usando herança.

Em termos de usar a herança em seu próprio código, você provavelmente não a usará com frequência, principalmente no começo e em pequenos projetos. É uma perda de tempo usar objetos e herança apenas por causa dela quando você não precisa deles. Mas à medida que suas bases de código aumentam, é mais provável que você encontre uma necessidade para isso. Se você estiver começando a criar vários objetos com recursos semelhantes, criar um tipo de objeto genérico para conter toda a funcionalidade compartilhada e herdar esses recursos em tipos de objetos mais especializados pode ser conveniente e útil.

Nota: Por causa da maneira como o JavaScript funciona, com a cadeia de protótipos, etc., o compartilhamento de funcionalidade entre objetos é frequentemente chamado de **delegação**. Os objetos especializados delegam a funcionalidade a um tipo de objeto genérico.

Ao usar a herança, você é aconselhado a não ter muitos níveis de herança, e manter um controle cuidadoso de onde você define seus métodos e propriedades. É possível começar a escrever código que modifica temporariamente os protótipos dos objetos do navegador interno, mas você não deve fazer isso a menos que tenha um bom motivo. Demasiada herança pode levar a confusão sem fim, e dor infinita quando você tenta depurar esse código.

Em última análise, os objetos são apenas outra forma de reutilização de código, como funções ou loops, com seus próprios papéis e vantagens específicos. Se você estiver criando um monte de variáveis e funções relacionadas e quiser rastreá-las todas juntas e empacotá-las perfeitamente, um objeto é uma boa ideia. Objetos também são muito úteis quando você quer passar uma coleção de dados de um lugar para outro. Ambas as coisas podem ser alcançadas sem o uso de construtores ou herança. Se você precisa apenas de uma única instância de um objeto, provavelmente é melhor usar apenas um literal de objeto e certamente não precisa de herança.