Object

# Objeto

Los objetos de JavaScript nunca están realmente vacíos porque pueden recoger miembros de la cadena de prototipos. A veces eso importa. Por ejemplo, supongamos que estás escribiendo un programa que cuenta el número de apariciones de cada palabra en un texto. Podemos utilizar el método toLowerCase para normalizar el texto a minúsculas, y luego utilizar el método split con una expresión regular para producir una matriz de palabras. A continuación, podemos recorrer las palabras y contar el número de veces que aparece cada una:

var i;  
var word;  
var text =  
 "This oracle of comfort has so pleased me, " +  
 "That when I am in heaven I shall desire " +  
 "To see what this child does, " +  
 "and praise my Constructor.";  
  
var words = text.toLowerCase( ).split(/[\s,.]+/);  
var count = {};  
for (i = 0; i < words.length; i += 1) {  
 word = words[i];  
 if (count[word]) {  
 count[word] += 1;  
 } else {  
 count[word] = 1;  
 }  
}

Si nos fijamos en los resultados, count['this'] es 2 y count.heaven es 1, pero count.constructor contiene una cadena de aspecto alocado. La razón es que el objeto count hereda de Object.prototype, y Object.prototype contiene un miembro llamado constructor cuyo valor es Object. El operador +=, al igual que el operador +, realiza la concatenación en lugar de la suma cuando sus operandos no son números. Object es una función, por lo que +=la convierte de algún modo en una cadena y concatena un 1 a su extremo.

Podemos evitar problemas como éste del mismo modo que evitamos problemas con for in: comprobando la pertenencia con el método hasOwnProperty o buscando tipos específicos. En este caso, nuestra prueba de veracidad de count[word] no era lo suficientemente específica. Podríamos haber escrito en su lugar

if (typeof count[word] === 'number') {