Memoization

# Memoización

Las funciones pueden utilizar objetos para recordar los resultados de operaciones anteriores, lo que permite evitar trabajo innecesario. Esta optimización se denomina *memoization*. Los objetos y matrices de JavaScript son muy prácticos para esto.

Supongamos que queremos una función recursiva para calcular números de Fibonacci. Un número de Fibonacci es la suma de los dos números de Fibonacci anteriores. Los dos primeros son 0 y 1:

var fibonacci = function (n) {  
 return n < 2 ? n : fibonacci(n − 1) + fibonacci(n − 2);  
};  
  
for (var i = 0; i <= 10; i += 1) {  
 document.writeln('// ' + i + ': ' + fibonacci(i));  
}  
  
// 0: 0  
// 1: 1  
// 2: 1  
// 3: 2  
// 4: 3  
// 5: 5  
// 6: 8  
// 7: 13  
// 8: 21  
// 9: 34  
// 10: 55

Esto funciona, pero hace mucho trabajo innecesario. La función fibonacci se llama 453 veces. Nosotros la llamamos 11 veces, y ella se llama a sí misma 442 veces para calcular valores que probablemente ya se han calculado recientemente. Si *memoize* la función, podemos reducir significativamente su carga de trabajo.

Guardaremos nuestros resultados memoizados en una matriz memoque podemos ocultar en un cierre. Cuando se llama a nuestra función, primero mira si ya conoce el resultado. Si lo conoce, puede devolverlo inmediatamente:

var fibonacci = (function ( ) {  
 var memo = [0, 1];  
 var fib = function (n) {  
 var result = memo[n];  
 if (typeof result !== 'number') {  
 result = fib(n − 1) + fib(n − 2);  
 memo[n] = result;  
 }  
 return result;  
 };  
 return fib;  
}( ));

Esta función devuelve los mismos resultados, pero sólo se llama 29 veces. La llamamos 11 veces. Se llamó a sí misma 18 veces para obtener los resultados previamente memorizados.

Podemos generalizar esto creando una función que nos ayude a hacer funciones memoizadas. La función memoizer tomará una matriz inicial memo y la función formula. Devuelve una función recur que gestiona el almacén de memo y que llama a la función formula cuando es necesario. Pasamos la función recur y los parámetros de la función a la función formula:

var memoizer = function (memo, formula) {  
 var recur = function (n) {  
 var result = memo[n];  
 if (typeof result !== 'number') {  
 result = formula(recur, n);  
 memo[n] = result;  
 }  
 return result;  
 };  
 return recur;  
};

Ahora podemos definir fibonacci con el memoizer, proporcionando la matriz inicial memo y la función formula:

var fibonacci = memoizer([0, 1], function (recur, n) {  
 return recur(n − 1) + recur(n − 2);  
});

Ideando funciones que produzcan otras funciones, podemos reducir significativamente la cantidad de trabajo que tenemos que hacer. Por ejemplo, para producir una función factorial memoizante, sólo tenemos que proporcionar la fórmula factorial básica:

var factorial = memoizer([1, 1], function (recur, n) {  
 return n \* recur(n − 1);  
});