Prototypal

# Prototípico

En un patrón puramente prototípico, prescindimos de las clases. En su lugar, nos centramos en los objetos. La herencia prototípica es conceptualmente más sencilla que la herencia clásica: un objeto nuevo puede heredar las propiedades de un objeto antiguo. Esto quizá no te resulte familiar, pero es muy fácil de entender. Empiezas haciendo un objeto útil. Luego puedes hacer muchos más objetos que sean como ése. El proceso de clasificación de descomponer una aplicación en un conjunto de clases abstractas anidadas puede evitarse por completo.

Empecemos utilizando un literal de objeto para crear un objeto útil:

var myMammal = {  
 name : 'Herb the Mammal',  
 get\_name : function ( ) {  
 return this.name;  
 },  
 says : function ( ) {  
 return this.saying || '';  
 }  
};

Una vez que tengamos un objeto que nos guste, podemos hacer más instancias con el método Object.create del [Capítulo 3](ch03.html). Luego podemos personalizar las nuevas instancias:

var myCat = Object.create(myMammal);  
myCat.name = 'Henrietta';  
myCat.saying = 'meow';  
myCat.purr = function (n) {  
 var i, s = '';  
 for (i = 0; i < n; i += 1) {  
 if (s) {  
 s += '-';  
 }  
 s += 'r';  
 }  
 return s;  
};  
myCat.get\_name = function ( ) {  
 return this.says() + ' ' + this.name + ' ' + this.says();  
};

Esto es *differential inheritance*. Al personalizar un nuevo objeto, especificamos las diferencias con el objeto en el que se basa.

A veces es útil que las estructuras de datos hereden de otras estructuras de datos. He aquí un ejemplo: Supongamos que estamos analizando un lenguaje como JavaScript oTEXen el que un par de llaves indica un ámbito. Los elementos definidos en un ámbito no son visibles fuera de él. En cierto sentido, un ámbito interno hereda de su ámbito externo. Los objetos de JavaScript representan muy bien esta relación. Se llama a la función blockcuando se encuentra una llave izquierda. La función parse buscará símbolos en scope, y aumentará scope cuando defina nuevos símbolos:

var block = function ( ) {  
  
// Remember the current scope. Make a new scope that  
// includes everything from the current one.  
  
 var oldScope = scope;  
 scope = Object.create(scope);  
  
// Advance past the left curly brace.  
  
 advance('{');  
  
// Parse using the new scope.  
  
 parse(scope);  
  
// Advance past the right curly brace and discard the  
// new scope, restoring the old one.  
  
 advance('}');  
 scope = oldScope;  
};