



JS | Prototypal inheritance



Objetivos

- ✓ Comprender qué es la herencia por prototypes
- ✓ Crear clases que puedan heredar métodos de otras clases.
- ✓ La propiedad `__proto__`

Objetos y constructores

En JavaScript podemos crear objetos a partir de funciones constructoras:

```
function Animal(name, owner) {  
    this.name = name;  
    this.owner = owner;  
}  
  
var myAnimal = new Animal('Mickey', 'Disney');  
console.log(myAnimal.name);    //=> 'Mickey'
```

También podríamos crear objetos con
object literal notation:

```
var myAnimal = {  
  name: 'Mickey', owner: 'Disney'  
}  
  
console.log(myAnimal.name);    //=> 'Mickey'
```



Si podemos crear objetos de ambas maneras,
¿por qué crearlos con un constructor?



The constructor function

En JavaScript, cuando llamamos a una función usando la keyword *new*, esta función es un constructor. El constructor es un método usado para crear e inicializar objetos usando la misma estructura.

Nos permite crear objetos instanciados.

Veamos un ejemplo

Práctica 1



- Cree un constructor *Item* con *name* y *price* como propiedades.
- Luego cree dos objetos e inicialícelos con algunos valores

Cómo funciona el operador *new*

```
function Animal(name, owner) {  
    this.name = name;  
    this.owner = owner;  
}  
  
var myAnimal = new Animal("Arya", "Josephine");
```

*Recuerda que **this** tiene un valor diferente dependiendo de quién llama la función a la que se hace referencia.*

Object prototype

Ahora, vamos a ver cuál es el prototype del objeto y cómo afecta a nuestros objetos. Veamos un ejemplo:




Comportamiento compartido a través del prototipo de objeto

Cada objeto de JavaScript tiene un atributo `prototype`. El atributo `prototype` también es un objeto.

Todos los objetos de JavaScript heredan sus propiedades y métodos de su `prototype`. Cuando creamos un objeto literal o un `new Object()`, hereda de un prototipo llamado `Object.prototype` (el antecesor de todos los objetos)





```
Elements Console Sources Network Timeline >> ⋮
top ⌵ [ ] Preserve log [x] Show all messages
> function Animal (name, owner) {
  this.name = name;
  this.owner = owner;
}

var myAnimal = new Animal("Arya", "Josephine");
< undefined

> myAnimal
< ▼ Animal ⓘ
  name: "Arya"
  owner: "Josephine"
  ▼ __proto__: Object
    ▶ constructor: function Animal(name, owner)
    ▼ __proto__: Object
      ▶ __defineGetter__: function __defineGetter__()
      ▶ __defineSetter__: function __defineSetter__()
      ▶ __lookupGetter__: function __lookupGetter__()
      ▶ __lookupSetter__: function __lookupSetter__()
      ▶ constructor: function Object()
      ▶ hasOwnProperty: function hasOwnProperty()
      ▶ isPrototypeOf: function isPrototypeOf()
      ▶ propertyIsEnumerable: function propertyIsEnumerable()
      ▶ toLocaleString: function toLocaleString()
      ▶ toString: function toString()
      ▶ valueOf: function valueOf()
      ▶ get __proto__: function __proto__()
      ▶ set __proto__: function __proto__()

> |
```

Object prototype

Un prototipo de objeto es un objeto que contiene el constructor de la instancia, los métodos y propiedades del prototipo y el prototipo de todos los objetos de los que hereda. De esta manera, todas las instancias comparten el mismo comportamiento, ya que comparten el mismo prototipo.



Práctica 2

- Cree un método *CalculatePrice* para el constructor *Item*.
- Este método devolverá el *price* del objeto por defecto.
- Si el *name* del *Item* es *fruit*, devolverá el precio del *Item* menos el 5%.



ayuda para comenzar

```
function Item(name, price){
    this.name = name;
    this.price = price;
}

Item.prototype.calculatePrice = function() {
    /* ... */
}

var ball = new Item('soccer ball', 15);
ball.calculatePrice();
// => 15

var fruit = new Item('fruit', 10);
fruit.calculatePrice();
// => 9.5
```



Herencia

Function.prototype.call()

Deberíamos saber que todas las funciones de JavaScript son métodos de objeto. El método `call()` es un método de función de JavaScript predefinido.

Se puede utilizar para invocar (llamar) a una función con un *owner object* como primer argumento (parámetro) seguido de argumentos proporcionados individualmente.



Cómo se realiza la herencia en JavaScript - Heredar prototipo

Una vez que cambiamos el prototipo padre, el prototipo hijo también debe cambiar. ¿Cómo podemos hacer esto?

Establecer un nuevo tipo de objeto

Ahora tenemos una conexión directa entre nuestros objetos Animal y Dog a través del prototipo Animal, por lo que si el prototipo Animal cambia, también lo hará Dog. Pero todavía hay un último paso que debemos hacer para completar la herencia real en Javascript.





Al verificar el objeto myDog (en el ejemplo anterior), que es una instancia del constructor Dog, podemos ver que el prototipo es un objeto prototipo Animal. El problema es que, dado que no hay un constructor para los objetos Dog en el objeto prototipo Animal, tomará el constructor Animal como propio.

Vamos a arreglar la función del constructor Dog como el constructor predeterminado de los objetos Dog.



Object.create()

Este método devolverá un nuevo objeto con el objeto prototipo y las propiedades especificadas.

La propiedad `__proto__`

Hasta ahora siempre nos referíamos al prototipo como un objeto almacenado dentro de cada objeto JavaScript. Por lo tanto, `__proto__` es la propiedad que tiene cada objeto y almacena este prototipo.



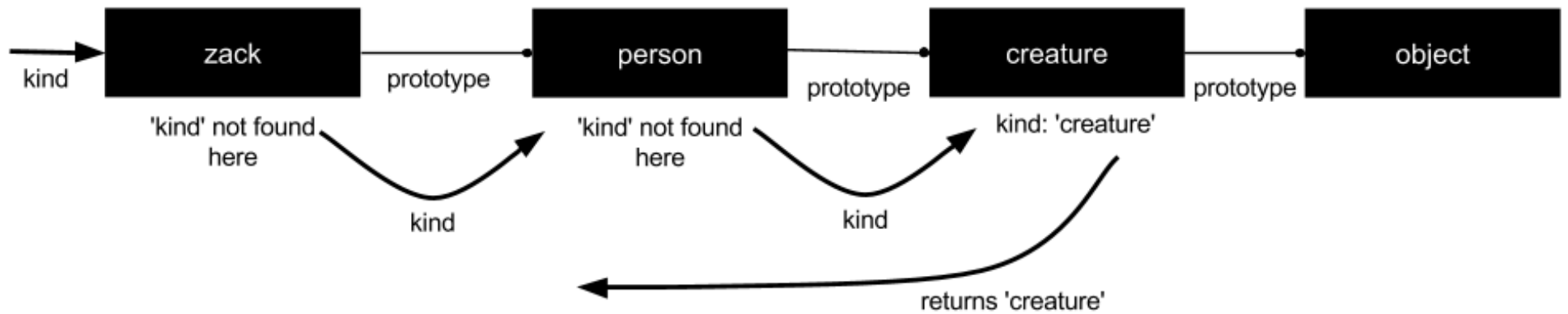


Cadenas de prototipo

Solo hay una construcción cuando se trata de herencia en JavaScript: objetos.

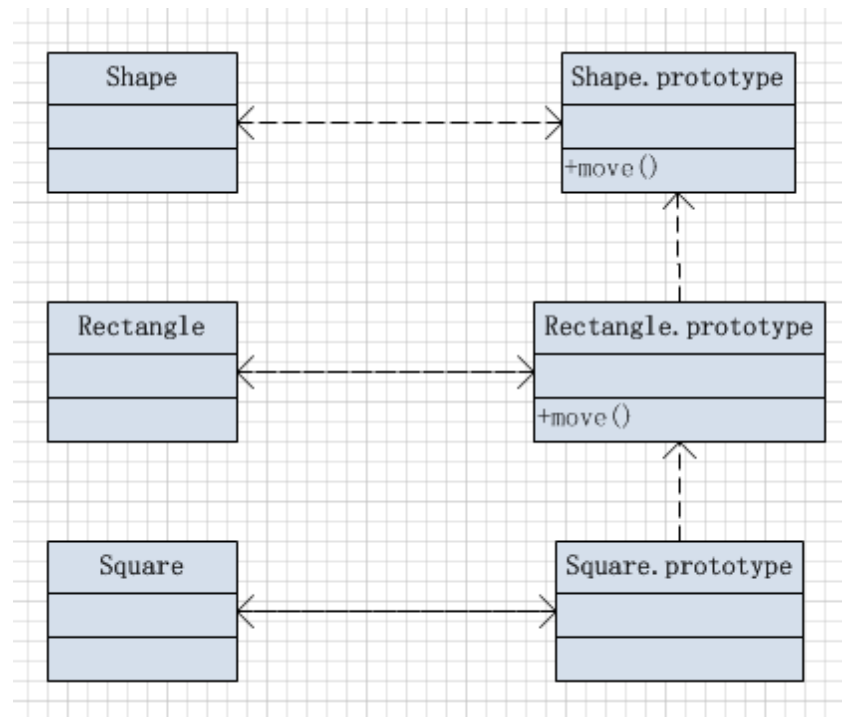
Todos los objetos tienen enlaces internos a otros objetos dentro de la propiedad `__proto__` - los llamamos prototipos de "otros objetos"; y ese objeto prototipo tendrá un prototipo heredado propio. Esto continúa hasta que encontramos un objeto con un prototipo nulo (lo que significa que no tiene propiedad `__proto__`).

Cadenas de prototipo



Prototipos como punteros

La forma más fácil de ver prototipos en JavaScript es como punteros de otros objetos guardados en el objeto actual. Miremos el siguiente diagrama:



Acceso a los métodos y atributos de un prototipo.

A veces, necesitamos acceder a los métodos y atributos de nuestro padre.

Por ejemplo, imagine que tenemos un método en el prototipo de Animal que sobrescribimos en un objeto Animal específico, pero en realidad necesitamos ejecutar el método prototipo.

Veamos un ejemplo:



Resumiendo: aprendimos...

- ✓ cómo construir un constructor, cómo el nuevo operador crea los objetos y cómo funciona esto dentro de la función del constructor.
- ✓ sobre el prototipo de objeto, una presencia de objeto en cada objeto JavaScript, lo que hace posible compartir métodos y atributos entre objetos instanciados del mismo constructor. El prototipo de objeto también nos permite usar la herencia a través de la cadena de prototipos, trabajando como punteros entre los objetos contruidos a través de la propiedad `__proto__`.