

Objetos

En Javascript, existe un tipo de dato llamado objeto . Una primera forma de verlo, es como una variable especial que puede contener más variables en su interior. De esta forma, tenemos la posibilidad de organizar múltiples variables de la misma temática en el interior de un objeto.

```
const producto = { nombre:  
  'manzana', categoria: 'frutas',  
  precio: 1.99 }  
  
console.log(producto);
```

Los objetos en JavaScript son colecciones dinámicas de pares clave-valor. La clave es siempre una cadena y debe ser única en la colección. El valor puede ser una primitiva, un objeto o incluso una función.

Podemos acceder a una propiedad usando el punto o la notación cuadrada.

```
console.log(producto.nombre);  
// "manzana"  
console.log(producto["nombre"]);  
// "manzana"
```

Aquí hay un ejemplo de donde valor es el otro objeto.

```
const producto = {  
  nombre: 'manzana',  
  categoria: 'frutas',  
  precio: 1.99,  
  nutrientes :  
    { carbohidratos: 0.95,  
      grasas: 0.3,  
      proteina: 0.2 }  
}
```

El valor de la propiedad carbohidratos es un nuevo objeto. Aquí es como nosotros podemos acceder a la propiedad carbohidratos .

```
console.log(producto.nutrientes.carbohidratos); // 0.95
```

Object.create

A continuación, veamos cómo implementar objetos con comportamiento, objetos orientados a objetos. JavaScript tiene lo que se llama el sistema prototípico que permite compartir el comportamiento entre objetos. La principal idea es crear un objeto llamado prototípico con un comportamiento común y luego usarlo cuando se crean nuevos objetos.

El sistema prototípico nos permite crear objetos que heredan comportamientos de los otros objetos. Vamos a crear un prototípico objeto que nos permite agregar productos y obtener el precio total de un carro de compras.

```
const prototipoCarrito = {  
  agregarProducto:  
    function(producto){  
      if(!this.productos){  
        this.productos = [producto]  
      } else {  
        this.productos.push(producto);  
      }  
    }, obtenerPrecioTotal: function(){  
      return  
        this.productos.reduce((total, p) =>  
          total + p.precio, 0);  
    }  
}
```

Vea que esta vez el valor de la propiedad agregarProducto es una función. También podemos escribir el previo objeto usando una forma corta llamada sintaxis del método abreviado.

```
const prototipoCarrito = {  
  agregarProducto(producto)  
  /* código */,  
  obtenerPrecioTotal()/* código */ }
```

Él prototipoCarrito es un prototípico objeto que mantiene un comportamiento común representado por dos métodos, agregarProducto y obtenerPrecioTotal. Puede ser usado para construir otros objetos heredando este comportamiento.

```
const carrito = Object.create(prototipoCarrito);  
carrito.agregarProducto({nombre: 'naranja', precio: 1.25});  
carrito.agregarProducto({nombre: 'limón', precio: 1.75});  
console.log(carrito.obtenerPrecioTotal()); // 3
```

El objeto carrito tiene prototipoCarrito como prototípico. Hereda el comportamiento de él. Carrito tiene una propiedad escondida que apunta al objeto prototípico. Cuando usamos un método en un objeto, ese método se busca primero en el objeto mismo en lugar de en su prototípico.

this

Vea que estamos usando una palabra clave especial llamada this para acceder y modificar los datos en el objeto.

Recuerda que las funciones son unidas independientes de comportamiento en JavaScript. No son necesariamente parte de un objeto. Cuando lo están, necesitamos tener una referencia que permita a la función acceder a otros miembros del mismo objeto. this es el contexto de la función. Da acceso a las otras propiedades.

Datos

Quizás te preguntes por qué no hemos definido e inicializado la propiedad de productos propiedad en el objeto prototípico en sí.

No deberíamos hacer eso. Los prototípicos deben usarse para compartir comportamientos, no datos. Compartir datos conducirá a tener los mismos productos en varios objetos del carro.

Considera el siguiente código:

```
const prototipoCarrito = {  
  productos: [], agregarProducto:  
    function(producto){  
      this.productos.push(producto);  
    }, obtenerPrecioTotal: function(){  
      return  
        this.productos.reduce((total, p) =>  
          total + p.precio, 0);  
    }  
}
```

Ambos carro1 y carro2 objetos heredan el mismo comportamiento del prototipoCarrito también comparten el mismo data. Los prototípicos que deberíamos usar comparten comportamiento y no data.

Close

El prototípico sistema no es una forma común de construir objetos. Los desarrolladores están más familiarizados en construir objetos fuera de las clases.

La sintaxis de la clase permite una forma de crear objetos que comparten un comportamiento común. Todavía crea el mismo prototípico detrás de escena, pero la sintaxis es más clara y también evita el problema anterior relacionado con los datos. La clase ofrece un lugar específico para definir los datos distintos para cada objeto.

Aquí está el mismo objeto creado usando la sintaxis de la clase azúcar (sugar syntax):

```
class Carrito{  
  constructor(){  
    this.productos = [];  
  }  
  agregarProducto(producto){  
    this.productos.push(producto);  
  }  
  obtenerPrecioTotal(){  
    return this.productos.reduce((total, p) => total + p.precio, 0);  
  }  
}  
  
const carro = new Carrito();  
carro.agregarProducto({nombre: 'naranja', precio: 1.25});  
carro.agregarProducto({nombre: 'limón', precio: 1.75});  
console.log(carro.obtenerPrecioTotal()); // 3
```

Observa que la clase tiene un método constructor que inicializa esos datos distintos para cada objeto nuevo. Los datos del constructor no se comparten entre instancias.

Para crear una nueva instancia, usamos la palabra clave new.

Creo que la sintaxis de la clase es más clara y familiar para la mayoría de los desarrolladores. Sin embargo, hace algo similar, crea un prototípico con todos los métodos y lo usa para definir nuevos objetos. Se puede acceder al prototípico con Carrito.prototype.

Resulta que el sistema prototípico es lo suficientemente flexible como para permitir la sintaxis de la clase. Por tanto, el sistema de clases se puede simular utilizando el sistema de prototípicos.

Close

El prototípico sistema no es una forma común de construir objetos. Los desarrolladores están más familiarizados en construir objetos fuera de las clases.

La sintaxis de la clase permite una forma de crear objetos que comparten un comportamiento común. Todavía crea el mismo prototípico detrás de escena, pero la sintaxis es más clara y también evita el problema anterior relacionado con los datos. La clase ofrece un lugar específico para definir los datos distintos para cada objeto.

Aquí está el mismo objeto creado usando la sintaxis de la clase azúcar (sugar syntax):

```
class Carrito{  
  constructor(){  
    this.productos = [];  
  }  
  agregarProducto(producto){  
    this.productos.push(producto);  
  }  
  obtenerPrecioTotal(){  
    return this.productos.reduce((total, p) => total + p.precio, 0);  
  }  
}  
  
const carro = new Carrito();  
carro.agregarProducto({nombre: 'naranja', precio: 1.25});  
carro.agregarProducto({nombre: 'limón', precio: 1.75});  
console.log(carro.obtenerPrecioTotal()); // 3
```

Observa que la clase tiene un método constructor que inicializa esos datos distintos para cada objeto nuevo. Los datos del constructor no se comparten entre instancias.

Para crear una nueva instancia, usamos la palabra clave new.

Creo que la sintaxis de la clase es más clara y familiar para la mayoría de los desarrolladores. Sin embargo, hace algo similar, crea un prototípico con todos los métodos y lo usa para definir nuevos objetos. Se puede acceder al prototípico con Carrito.prototype.

Resulta que el sistema prototípico es lo suficientemente flexible como para permitir la sintaxis de la clase. Por tanto, el sistema de clases se puede simular utilizando el sistema de prototípicos.