Curso de Javascript

Unidad Didáctica 09: Herencia

http://cursosdedesarrollo.com/

Ayuntamiento de Vitoria-Gasteiz
Vitoria-Gasteizko Udala

Índice de contenidos

http://cursosdedesarrollo.com/

Introducción

JavaScript en un lenguaje orientado a objetos basado en prototipos, en lugar de estar basado en clases

http://cursosdedesarrollo.com/

Introducción

Debido a esta básica diferencia, es menos evidente entender cómo JavaScript nos permite crear herencia entre objetos, y heredar las propiedades y sus valores

http://cursosdedesarrollo.com/

Prototype

Todos los objetos de JavaScript enlazan con un objeto prototipo del que heredan todas sus propiedades

Los objetos creado a través de literales, están enlazados con Object.prototype, un objeto estándar incluido en JavaScript

http://cursosdedesarrollo.com/

Prototype

Cuando creamos un objeto nuevo, tenemos la posibilidad de seleccionar cuál será su prototipo

El mecanismo que JavaScript proporciona para hacer esto es desordenado y complejo, pero se puede simplificar de manera significativa

http://cursosdedesarrollo.com/

Prototype

Vamos a añadir un método de creación a nuestro objeto

El método create crea un nuevo objeto que utiliza un objeto antiguo como su prototipo

http://cursosdedesarrollo.com/

Prototype

```
// Shape - superclass

function Shape() {

this.x = 0;

this.y = 0;

}
```

http://cursosdedesarrollo.com/

Prototype

```
Shape.prototype.move = function(x, y) {

    this.x += x;

    this.y += y;

    console.info("Shape moved.");

};
```

http://cursosdedesarrollo.com/

Prototype

```
// Rectangle - subclass

function Rectangle() {
  Shape.call(this); //call super constructor.

}

Rectangle.prototype =
Object.create(Shape.prototype);
```

http://cursosdedesarrollo.com/

Prototype

```
var rect = new Rectangle();

rect instanceof Rectangle  // true.

rect instanceof Shape      // true.

rect.move(); // Outputs, "Shape moved."
```

http://cursosdedesarrollo.com/

Prototype

Para los navegadores que no soportan la función create, podemos extender el objeto de JavaSript Object para incluir esta funcionalidad:

```
if (typeof Object.create !== 'function') {

    Object.create = function (o) {
        var F = function () {};
        F.prototype = o;
        return new F();
    };

}

var another_stooge = Object.create(stooge);
```

http://cursosdedesarrollo.com/

Prototype

El prototipo enlazado no se ve afectado por las modificaciones. Si realizamos cambios en un objeto, el objeto prototipo no se ve afectado

http://cursosdedesarrollo.com/

Prototype

El enlace de los prototipos es únicamente utilizado cuando accedemos a los datos

Si intentamos acceder al valor de una propiedad, y esa propiedad no existe en el objeto, entonces JavaScript va a intentar obtener ese valor del prototipo del objeto

http://cursosdedesarrollo.com/

Prototype

Y si ese objeto tampoco dispone de la propiedad, lo intentará obtener de sucesivos prototipos, hasta que finalmente se encuentre con Object.prototype

Si la propiedad no existe en ninguno de los prototipos, entonces el valor devuelto es undefined

http://cursosdedesarrollo.com/

Prototype

La relación de prototipos es dinámica. Si nosotros añadimos una nueva propiedad a un prototipo, entonces esta propiedad estará inmediatamente accesible para el resto de prototipos que estén basados en ese prototipo:

```
stooge.profession = 'actor';

another_stooge.profession      // 'actor'
```

http://cursosdedesarrollo.com/

Clases vs Prototipos

Los lenguajes orientados a objetos basados en clases como Java o C++, se basan en el concepto de dos entidades distintas: la clase y las instancias.

- Una clase define todas las propiedades que caracteriza a una serie de objetos. La clase es algo abstracto, no como las instancias de los objetos que describe. Por ejemplo, una clase Empleado, puede representar un conjunto concreto de empleados.

- Una instancia, en cambio, es una representación concreta de esa clase. Por ejemplo, Victoria puede ser una instancia concreta de la clase Empleado, es decir, representa de manera concreta a un empleado. Una instancia tiene exactamente las mismas propiedades que la clase padre (Ni más, ni menos).

http://cursosdedesarrollo.com/

Clases vs Prototipos

Un lenguaje basado en prototipos, como JavaScript, no hace esta distinción: simplemente maneja objetos

Este tipo de lenguajes tiene la noción de objetos prototipo, objetos usados como plantilla para obtener las propiedades iniciales de un objeto

http://cursosdedesarrollo.com/

Clases vs Prototipos

Cualquier objeto puede especificar su propias propiedades, tanto en el momento que los creamos como en tiempo de ejecución

Además, cualquier objeto puede asociarse como prototipo a otro objeto, permitiendo compartir todas sus propiedades

http://cursosdedesarrollo.com/

Definiendo una Clase

Un lenguaje basado en clases, definimos la clase de manera independiente

En esta definición, especificamos los constructores, que son utilizamos para crear las instancias de las clases

http://cursosdedesarrollo.com/

Definiendo una Clase

Un método constructor, puede especificar los valores iniciales de una instancia de una clase, y realizar las operaciones necesarias a la hora de crear el objeto

Utilizamos el operador new, conjuntamente con el nombre del constructor, para crear nuevas instancias

http://cursosdedesarrollo.com/

Definiendo una Clase

JavaScript sigue un modelo similar, pero no separa la definición de las propiedades del constructor

En este caso, definimos una función constructora para crear los objetos con un conjunto inicial de propiedades y valores

http://cursosdedesarrollo.com/

Definiendo una Clase

Cualquier función de JavaScript puede ser utilizada como constructor. Utilizamos el operador new, conjuntamente con el nombre de la función constructora, para crear nuevas instancias

http://cursosdedesarrollo.com/

Subclases y Herencia

En un lenguaje basado en clases, es posible crear estructura de clases a través de su definición

En esta definición, podemos especificar que la nueva clase es una subclase de una clase que ya existe

http://cursosdedesarrollo.com/

Subclases y Herencia

Esta subclase, hereda todas las propiedades de la superclase, y además puede añadir o modificar las propiedades heredadas

http://cursosdedesarrollo.com/

Subclases y Herencia

JavaScript implementa una herencia que nos permite asociar un objeto prototipo con una función constructora

De esta manera, el nuevo objeto hereda todas las propiedades del objeto prototipo

http://cursosdedesarrollo.com/

Diferencias con un lenguaje basado en clases

La siguiente tabla muestra un pequeño resumen de las diferencias entre un lenguaje basado en clases, como Java, y un lenguaje basado en prototipos, como JavaScript

http://cursosdedesarrollo.com/

Diferencias con un lenguaje basado en clases

Basado en clases (Java)	Basado en prototipos (JavaScript)
Clase e instancia son dos entidades diferentes	Todos los objetos son instancias
Las clases se definen de manera explícita, y se instancias a través de su método constructor.	Las clases se definen y crean con las funciones constructoras.
Un objeto se instancia con el operador `new`.	Un objeto se instancia con el operador `new`.
La estructura de clases se crea utilizando la definición de clases.	La estructura de clases se crea asignando un objeto como prototipo.
La herencia de propiedades se realiza a través de la cadena de clases.	La herencia de propiedades se realiza a través de la cadena de prototipos.
La definición de clases especifica todas las propiedades de una instancia de una clase. No se pueden añadir propiedades en tiempo de ejecución.	La función constructora o el prototipo especifican unas propiedades iniciales. Se pueden añadir o eliminar estas propiedades en tiempo de ejecución, en un objeto concreto o a un conjunto de objetos.

http://cursosdedesarrollo.com/

Creando la Herencia

Veamos como se implementa esta herencia en JavaScript, a través de un simple ejemplo. Queremos implementar la siguiente estructura:

- Un Empleado se define con las propiedades nombre (cuyo valor por defecto es una cadena vacía), y un departamento (cuyo valor por defecto es "General").

- Un Director está basado en Empleado. Añade la propiedad informes (cuyo valor por defecto es un array vacío).

- Un Trabajador está basado también en Empleado. Añade la propiedad proyectos (cuyo valor por defecto es un array vacío).

- Un Ingeniero está basado en Trabajador. Añade la propiedad maquina (cuyo valor por defecto es una cadena vacía) y sobreescribe la propiedad departamento con el valor "Ingeniería".

Creando la Herencia

```
function Empleado (nombre, departamento) {

this.nombre = nombre || "";

this.departamento = departamento || "General";

}

function Director (nombre, departamento, informes) {

this.base = Empleado;

this.base(nombre, departamento);

this.informes = informes || [];

}

Director.prototype = new Empleado;
```

http://cursosdedesarrollo.com/

Creando la Herencia

```
function Obrero (nombre, departamento, proyectos) {

    this.base = Empleado;

    this.base(nombre, departamento);

    this.proyectos = proyectos || [];

}

Obrero.prototype = new Empleado;
```

http://cursosdedesarrollo.com/

Creando la Herencia

```
function Ingeniero (nombre, proyectos, maquina) {

    this.base = Obrero;

    this.base(nombre, "Ingeniería", proyectos);

    this.maquina = maquina || "";

}

Ingeniero.prototype = new Obrero;
```

http://cursosdedesarrollo.com/

Creando la Herencia

Supongamos que queremos crear un nuevo Ingeniero de la siguiente manera:

```
var arkaitz = new Ingeniero("Garro, Arkaitz",
                            ["xhtml", "javascript", "html5"],
                            "Chrome");
```

http://cursosdedesarrollo.com/

Creando la Herencia

JavaScript sigue el siguiente proceso:

1. El operador new creará un objeto genérico y asignará a la propiedad __proto__ el valor de Ingeniero.prototype.

2. El operador new pasará el nuevo objeto creado al constructor de Ingeniero, como valor de la palabra reservada this.

3. El constructor crea una nueva propiedad llamada base para este objeto, y le asigna el valor del constructor Obrero. Esto convierte el constructor de Obreroen un método del objeto Ingeniero. El nombre baseno hace referencia a ninguna palabra reservada, es simplemente para hacer referencia al padre.

4. El constructor llama al método base, pasando como argumentos dos de sus argumentos ("Garro, Arkaitz" y ["xhtml", "javascript", "html5"]), además de la cadena de caracteres "Ingeniería". Indicar este parámetro fijo, hace que todos los objetos creados de tipo Ingeniero, tengan el mismo valor para la propiedad departamento, sobreescribiendo la el valor original de Empleado.

5. Al llamar al método base, JavaScript asocia la palabra reservada this al objeto creado en el paso 1. En consecuencia, la función Obrero pasa los valores "Garro, Arkaitz" y ["xhtml", "javascript", "html5"] al constructor de Empleado. Cuando se completa este paso, la función Obrero asigna el valor de los projectos a su propiedad.

6. Una vez finalizado el método base, el constructor de Ingeniero asigna el valor Chrome a la propiedad maquina.

7. Una vez finalizado el constructor, JavaScript asigna el nuevo objeto a la variable arkaitz.

http://cursosdedesarrollo.com/

Determinando la relación entre instancias

La búsqueda de propiedades en JavaScript comienza en las propias propiedades del objeto, y si este nombre de propiedad no se encuentra, consulta las propiedades del objeto especial __proto__. Este proceso se realiza de manera recursiva

http://cursosdedesarrollo.com/

Determinando la relación entre instancias

La propiedad especial __proto__ se define cuando un objeto es construido: su valor corresponde con la propiedad prototype del constructor. Así, la expresión new Foo() crea un objeto con la propiedad __proto__ == Foo.prototype

http://cursosdedesarrollo.com/

Determinando la relación entre instancias

En consecuencia, los cambios producidos en Foo.prototype alteran la búsqueda de propiedades de todos los objetos creados con new Foo()

Determinando la relación entre instancias

Todo objeto tiene una propiedad __proto__, así como una propiedad prototype. Por lo tanto, los objetos pueden relacionarse a través de esta propiedad. Un ejemplo:

```
var arkaitz = new Ingeniero("Garro, Arkaitz",
                            ["xhtml", "javascript", "html5"],
                            “Chrome");
```

http://cursosdedesarrollo.com/

Determinando la relación entre instancias

En este objeto, todas las siguientes sentencias se cumplen:

arkaitz.__proto__ == Ingeniero.prototype;

arkaitz.__proto__.__proto__ == Obrero.prototype;

arkaitz.__proto__.__proto__.__proto__ == Empleado.prototype;

arkaitz.__proto__.__proto__.__proto__.__proto__ == Object.prototype;

arkaitz.__proto__.__proto__.__proto__.__proto__.__proto__ == null;

http://cursosdedesarrollo.com/

Determinando la relación entre instancias

De esta manera, podemos counstruir una función apara saber si un objeto es una variable es una instancia de una objeto.

```
function instanceOf(object, constructor) {

while (object != null) {

if (object == constructor.prototype)
return true;

if (typeof object == 'xml') {

return constructor.prototype == XML.prototype;

}

object = object.__proto__;

}

return false;

}
```

http://cursosdedesarrollo.com/

Determinando la relación entre instancias

instanceOf (arkaitz, Engineer) // true

instanceOf (arkaitz, WorkerBee) // true

instanceOf (arkaitz, Employee) // true

instanceOf (arkaitz, Object) // true

instanceOf (arkaitz, SalesPerson) // false

http://cursosdedesarrollo.com/

Conclusiones

Hemos visto cómo manejar
la herencia en Javascript

http://cursosdedesarrollo.com/

Datos de Contacto

http://www.cursosdedesarrollo.com
info@cursosdedesarrollo.com

http://cursosdedesarrollo.com/

Licencia

David Vaquero Santiago

Esta obra está bajo una
Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional
Deribada de:
http://www.arkaitzgarro.com/javascript/
y
http://javiereguiluz.com/

http://cursosdedesarrollo.com/