APRENDERAPROGRAMAR.COM

CALL JAVASCRIPT. DIFERENCIAS CON APPLY. CONSTRUCTORES CON HERENCIA EN CADENA (PROTOTYPE CHAIN). EJEMPLOS (CU01150E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº50 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

CADENAS DE HERENCIA JAVASCRIPT

JavaScript permite la herencia a través de prototipos, pero todavía no hemos visto cómo podemos crear un objeto de un subtipo que pueda ser instanciado para que se inicialice portando información tanto del subtipo como del supertipo. Antes de abordar esta cuestión estudiaremos las funciones call y apply.

CALL JAVASCRIPT

La función call permite llamar a cualquier función JavaScript indicándole el objeto que actuará como this dentro de la función llamada, así como los parámetros adicionales que sean necesarios.

La sintaxis más básica es la siguiente:

```
function unaFuncion (par1, par2, ..., parN) {
// código
... }
unaFuncion.call (objetoQueActuaráComoThis, par1, par2, ... parN);
```

Veamos un ejemplo básico para comprender lo que significa y permite esta función. Escribe el código y comprueba el resultado.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function Profesor (nombre) { this.nombre = nombre || 'Nombre desconocido'; this.salarioBase = 1200; }
function saludar() { alert ('Hola, soy ' + this.nombre); }
function ejemploObjetos() {
var unProfesor = new Profesor('Carlos');
saludar();
saludar.call(unProfesor);
unProfesor.saludar(); //ERROR - NO PERMITIDO
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

El resultado de ejecución esperado es: Hola, soy undefined. >>> Hola, soy Carlos. >>> (Y un error: la línea unProfesor.saludar() no se ejecuta).

Analicemos paso a paso lo que hace el código:

© aprenderaprogramar.com, 2006-2029

Se define el tipo de objeto Profesor con los atributos nombre y salarioBase. Se define la función saludar() que carece de propiedades y que cuando se invoca muestra por pantalla un mensaje.

Se crea un objeto de tipo Profesor cuya propiedad nombre vale "Carlos".

Se llama a la función saludar() y como resultado se obtiene "Hola, soy undefined". ¿Por qué? Porque el objeto this en este caso es la propia función saludar y dicha función no tiene definido atributo nombre, por tanto al tratar de mostrar this.nombre muestra 'undefined'.

A continuación se llama a la función saludar indicando que el objeto unProfesor actuará como objeto this para la ejecución de la función. Como consecuencia, this.nombre vale "Carlos" y se muestra por pantalla el mensaje "Hola, soy Carlos".

El quid de la cuestión está en que cualquier función puede invocarse como método de cualquier objeto a través del método predefinido JavaScript call. En el ejemplo que hemos usado podemos destacar algunas cuestiones:

a) saludar() es lo mismo que saludar.call(). Al indicar saludar.call() y no pasar como parámetro ningún objeto, es la propia función saludar quien actúa como this.

b) La función saludar podría ser usada por diversos tipos de objetos que tuvieran un atributo name. La función actuaría así como método compartido por numerosos tipos de objetos, pudiendo estar fuera de la definición de los mismos y fuera de la cadena de herencia de los mismos.

Veamos ahora la sintaxis usando parámetros. Escribe este código y comprueba el resultado.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function Profesor (nombre) { this.nombre = nombre || 'Nombre desconocido'; this.salarioBase = 1200; }
function saludar(nombrePersona, modoSaludo) {
alert ('Hola, soy ' + this.nombre + ' y saludo a ' +nombrePersona+ ' con ' + modoSaludo); }
function ejemploObjetos() {
var unProfesor = new Profesor('Carlos');
saludar.call();
saludar.call(unProfesor, 'Ernesto', 'afecto');
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

El resultado esperado es: Hola, soy undefined y saludo a undefined con undefined. >>> Hola, soy Carlos y saludo a Ernesto con afecto.

En la invocación saludar.call(unProfesor, 'Ernesto', 'afecto'); el parámetro unProfesor indica quién va a actuar como objeto this en la ejecución de la función saludar, mientras que 'Ernesto' y 'afecto' son los parámetros que se le pasan a la función.

© aprenderaprogramar.com, 2006-2029

APPLY JAVASCRIPT

La función apply permite llamar a cualquier función JavaScript indicándole el objeto que actuará como this dentro de la función llamada, de la misma forma que con la función call. La diferencia de apply con call está en que los parámetros se pasan con un array en vez de separados por comas. De este modo apply consta exactamente de dos parámetros: el objeto que actuará como this y un array.

La sintaxis más básica es la siguiente:

```
function unaFuncion (par1, par2, ..., parN) {
// código
... }
unaFuncion.apply (objetoQueActuaráComoThis, arrayDeElementos);
```

Donde arrayDeElementos es un array. El array se puede haber declarado previamente, o bien declararse en el mismo momento de llamada de la función escribiendo [elemento1, elemento2, ..., etc.].

Veamos un ejemplo básico para comprender lo que significa y permite esta función. Escribe el código y comprueba el resultado.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function Profesor (nombre) { this.nombre = nombre || 'Nombre desconocido'; this.salarioBase = 1200; }
function saludar(nombrePersona, modoSaludo) {
alert ('Hola, soy ' + this.nombre + ' y saludo a ' +nombrePersona+ ' con ' + modoSaludo); }
function ejemploObjetos() { var unProfesor = new Profesor('Carlos');
saludar.apply();
var unArray = ['Christian', 'odio'];
saludar.apply(unProfesor, ['Ernesto', 'afecto']);
saludar.apply(unProfesor, unArray);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body></html>
```

El resultado de ejecución esperado es: Hola, soy undefined y saludo a undefined con undefined >>> Hola, soy Carlos y saludo a Ernesto con afecto >>> Hola, soy Carlos y saludo a Christian con odio.

Call y apply son dos funciones muy similares. ¿Cuándo usar una y cuándo usar otra? En muchos casos será indistinto usar una y otra. Sin embargo apply nos provee de la potencia de los arrays, lo que la hace interesante cuando por algún motivo necesitemos crear bucles que recorran los parámetros pasados en el array, o cuando simplemente no se conozca a priori el número de parámetros que deben pasarse porque se establezcan de forma dinámica.

© aprenderaprogramar.com, 2006-2029

CREAR HERENCIA CON JAVASCRIPT

Sabemos cómo hacer para que todos los objetos de un tipo hereden propiedades y métodos comunes de su prototipo, pero ¿cómo crear herencia entre dos tipos de objetos?

Partimos de un ejemplo: tenemos como subtipo ProfesorInterino con la propiedad mesesContrato y como supertipo Profesor con las propiedades institucion y salarioBase como muestra el siguiente código.

```
function Profesor (institucion) { this.institucion = institucion || 'Desconocida'; this.salarioBase = 1200; }

function ProfesorInterino(mesesContrato) { this.mesesContrato = mesesContrato || -1;}
```

Ahora nos planteamos que los Profesores Interinos son un tipo de profesor, y por tanto al crear un profesor interino queremos inicializarlo de modo que disponga de las propiedades y métodos de Profesor. Al crear un profesor interino queremos poder especificar el atributo institucion o, si no lo especificamos, poder acceder a la propiedad institucion obteniendo como valor 'Desconocida' ¿Cómo hacerlo?

Una primera aproximación puede ser esta:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

function Profesor (institucion) {
this.institucion = institucion || 'Desconocida';
this.salarioBase = 1200;
}

function ProfesorInterino(mesesContrato, institucion) {
Profesor.call(this, institucion);
this.mesesContrato = mesesContrato || -1;}

function ejemploObjetos() {
var unProfesorInterino = new ProfesorInterino(4, 'Universidad de Chapingo');
var msg = 'El objeto unProfesorInterino tiene ' + unProfesorInterino.mesesContrato + ' meses de contrato';
msg = msg + ' y pertenece a la institución '+unProfesorInterino.institucion;
alert(msg);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

El resultado de ejecución esperado es: El objeto unProfesorInterino tiene 4 meses de contrato y pertenece a la institución Universidad de Chapingo

© aprenderaprogramar.com, 2006-2029

Analicemos lo que ocurre. Hemos modificado la función ProfesorInterino para que además de las propiedades intrínsecas (mesesContrato) pueda recibir la institución, propiedad de los objetos tipo Profesor.

Con la invocación Profesor.call(this, institucion); estamos haciendo que se ejecute la función Profesor pasándole como this el objeto ProfesorInterino y como parámetros la institución. Al ejecutarse la función Profesor, el objeto ProfesorInterino pasa a tener todos los atributos y métodos de un Profesor ya que se ejecuta this.institucion = institucion || 'Desconocida'; y this.salarioBase = 1200.

De este modo, el objeto ProfesorInterino dispone de las propiedades y métodos propios de los objetos Profesor.

Pero, ¿realmente se comporta el objeto unProfesorInterino como los objetos de tipo Profesor? La respuesta es que no: a través de la llamada call hemos logrado que las propiedades y métodos declarados con this para Profesor estén disponibles para el profesor interino, pero ¿qué ocurre con las propiedades y métodos comunes de los objetos Profesor, es decir, aquellas propiedades y métodos que hayamos especificado en el prototipo de Profesor. Lo que ocurre es que no están disponibles para unProfesorInterino. Comprobémoslo. Escribe este código y comprueba los resultados.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">

function Profesor (institucion) { this.institucion = institucion || 'Desconocida';
this.salarioBase = 1200; }
Profesor.prototype.saludar = function() {alert('Hola trabajo en '+ this.institucion + ' y mi salario base es
'+this.salarioBase);}

function ProfesorInterino(mesesContrato, institucion) { Profesor.call(this, institucion);
this.mesesContrato = mesesContrato || -1;}

function ejemploObjetos() {
var unProfesor = new Profesor(); unProfesor.saludar();
var unProfesorInterino = new ProfesorInterino(4, 'Universidad de Chapingo');
var msg = 'El objeto unProfesorInterino tiene ' + unProfesorInterino.mesesContrato + ' meses de contrato';
msg = msg + ' y pertenece a la institución '+unProfesorInterino.institucion;
alert(msg);
unProfesorInterino.saludar();
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body></html>
```

El resultado esperado es: Hola trabajo en Desconocida y mi salario base es 1200 >> El objeto unProfesorInterino tiene 4 meses de contrato y pertenece a la institución Universidad de Chapingo >> **error** (la línea unProfesorInterino.saludar(); no se ejecuta).

Lo que comprobamos es que los objetos de tipo Profesor conocen el método saludar (que es un método común a todos los objetos de tipo Profesor definido a través de su prototipo), pero sin

© aprenderaprogramar.com, 2006-2029

embargo los objetos de tipo ProfesorInterino no conocen el método saludar, lo que demuestra que no es están comportando realmente como Profesor.

Para que los objetos de tipo ProfesorInterino dispongan de todo lo que dispone Profesor nos falta hacer que el prototipo de todo objeto ProfesorInterino sea un objeto Profesor. De este modo, cuando se busque el método saludar se buscará en primer lugar como método intrínseco de ProfesorInterino, al no encontrarse se buscará en el prototipo, que al ser un objeto Profesor sí podrá responder cuando se invoque el método saludar.

Para ello añadiremos la línea: ProfesorInterino.prototype = new Profesor();

Con esta línea ya los objetos de tipo ProfesorInterino conocen las propiedades y métodos comunes de Profesor.

Con esta línea ya se ejecuta la invocación unProfesorInterino.saludar() dando como resultado que se muestre por pantalla <<Hola trabajo en Universidad de Chapingo y mi salario base es 1200>>.

En resumen, para implementar una herencia completa y poder crear instancias pasando parámetros para generar objetos que hereden propiedades y métodos de un supertipo usaremos la invocación call al supertipo y además estableceremos que el prototipo es un objeto del supertipo.

CONSTRUCTORES CON HERENCIA

Con las herramientas que conocemos ya somos capaces de implementar cadenas de herencia con constructores que reciban parámetros de los supertipos. Supongamos que tenemos 4 clases en cadena y cada clase tiene una propiedad intrínseca. En la generación de un objeto del tipo inferior con herencia, tendríamos que pasarle 4 parámetros e invocar con 3 parámetros a su ascendiente. Luego el ascendiente será invocado con 2 parámetros y finalmente habrá una invocación con un parámetro.

A su vez, estableceremos como prototype de cada tipo de objeto a una instancia de su ascendiente.

ALTERNATIVA AL USO DE CALL

Podemos obtener los mismos efectos que con call usando esta construcción:

```
function unaFuncion (par1, par2, …, parN) {
this.fun = Superclase;
this.fun (param1, param2, …, paramN);
… }
```

Es decir, el código <<A>>: Profesor.call(this, institucion);

Da lugar al mismo efecto que el código <<B>>: this.fun = Profesor; this.fun(institucion);

En el primer caso (A) se ejecuta la función Profesor pasando como objeto this al objeto que lo llama y esto da lugar a que el objeto incorpore las propiedades y métodos del objeto llamado.

En el segundo caso (B) se define Profesor como un método propio del objeto y a continuación con la invocación this.prop(institucion) se ejecuta dicho método.

© aprenderaprogramar.com, 2006-2029

En principio call o apply resultan más eficientes, ya que nos ahorramos crear el método prop para posteriormente ejecutarlo, pero a efectos prácticos suele resultar indistinto y este tipo de invocaciones es frecuente encontrarlo cuando se revisa código JavaScript.

RESUMEN

La herencia en JavaScript no funciona como en otros lenguajes y en cierta medida más que de herencia propiamente dicha podríamos hablar de simulación de herencia. El siguiente cuadro resume las vías principales para generar herencia en JavaScript:

GENERADOR DE HERENCIA	DESCRIPCIÓN APRENDERAPROGRAMAR.COM
prototype.nombrePropiedadOMetodo	Define propiedades y métodos comunes que son compartidos por un tipo de objetos.
subtipo.prototype = new supertipo()	Crea herencia del supertipo pero no permite especificar los parámetros para los constructores de los supertipos.
nombreFuncion.call(objetoThis, par1, par2, ...) dentro de una función declarativa de tipo.	Crea herencia de propiedades y métodos declarados con this en los supertipos y permite inicializar objetos pasando parámetros para los subtipos y los supertipos, pero no da lugar a que se conozcan las propiedades y métodos compartidos del supertipo.
subtipo.prototype = new supertipo() + nombreFuncion.call(objetoThis, par1, par2, ...) dentro de una función declarativa de tipo	Permite simular una herencia completa y la instanciación de un objeto con todos los parámetros propios y de supertipos.

EJERCICIO 1

Crea un esquema de herencia que cumpla con estos requisitos:

a) Un Médico especialista tiene una especialidad y es un tipo de Médico.

b) Un Médico trabaja en un centro de trabajo y es un tipo de Persona.

c) Una Persona tiene un nombre y una nacionalidad. Como método común a todas las personas tenemos mostrarNacionalidad, que muestra un mensaje informando de la nacionalidad.

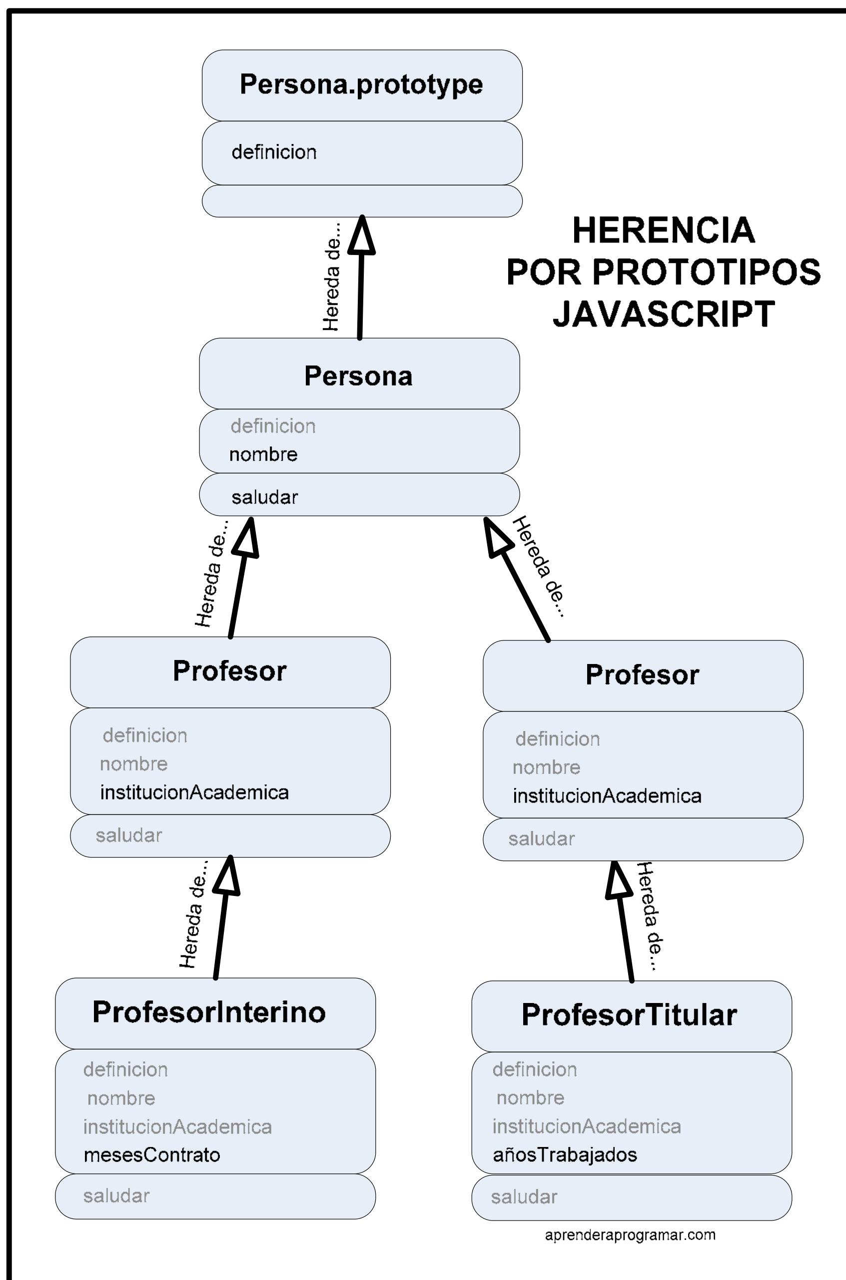
Se desea crear un objeto de tipo MedicoEspecialista pasándole como parámetros para su creación además de sus propiedades intrínsecas las propiedades que hereda de sus supertipos y sobre este objeto invocar el método mostrarNacionalidad(), que deberá ser reconocido por herencia.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

© aprenderaprogramar.com, 2006-2029

EJERCICIO 2

Crea un código que represente el siguiente esquema de herencia permitiendo instanciar los subtipos pasándole los parámetros necesarios para inicializar las propiedades de los supertipos. Crea un objeto ProfesorTitular profesorTitular1 al que le pases como parámetros 8 (años trabajados), Universidad de León (institución académica), Juan (nombre), e invoca el método saludar sobre este objeto.



Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01151E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206

© aprenderaprogramar.com, 2006-2029