¿Qué significa "this" en JavaScript? La palabra clave "this" explicada con ejemplos

Contexto this

Cuando se usa en una función, this simplemente apunta a un objeto al que está vinculado. Responde a la pregunta "**de dónde debería obtener algún valor o datos**":

```
function alert() {
  console.log(this.nombre + ' esta llamando');
}
```

En la función anterior, la palabra clave this se refiere a un objeto al que está vinculado **y obtiene la propiedad "nombre" de allí**.

Pero, ¿cómo saber a qué objeto está vinculada la función? ¿Cómo averiguas a qué se refiere this?

Para hacerlo, necesitamos echar un vistazo detallado a cómo las funciones están vinculadas a los objetos.

Tipos de enlaces en JavaScript

Por lo general, existen cuatro tipos de enlaces:

- Por defecto (predeterminada)
- implícito
- explícito
- de constructor

Enlace por defecto en JavaScript

Una de las primeras reglas que hay que recordar es que, si la función que contiene esta referencia this es una **función independiente**, esa función está vinculada al **objeto global**.

```
function alert() {
  console.log(this.nombre + ' esta llamando');
}
```

```
const nombre = 'Kingsley';
alert(); // Kingsley esta llamando
```

Función independiente

Como puedes ver, nombre() es una función independiente y no adjunta a un objeto, por lo que está vinculada al ámbito **global**. Como resultado, la referencia this.nombre se resuelve en la variable global **const nombre = 'Kingsley'**.

Sin Embargo, esta regla no se cumple si nombre() se definiera en modo estricto (strict mode):

```
function alert() {
  'use strict';
  console.log(this.nombre + ' esta llamando');
}

const nombre = 'Kingsley';
alert(); // TypeError: `this` is `undefined`
```

undefined en mode estricto

Cuando se establece como "strict mode", this es "undefined".

Enlace implícito en JavaScript

Otro escenario a tener en cuenta es si la función está adjunta a un objeto (su contexto) cuando se la llama.

De acuerdo con la regla de vinculación en JavaScript, una función puede usar un objeto como contexto solo si ese objeto está vinculado a él en la llamada. Esta forma de vinculación se conoce como vinculación implícita.

Esto es lo que quiero decir:

```
function alert() {
  console.log(this.edad + ' anyos');
}

const miObjecto = {
  edad: 22,
  alert: alert
}

miObjeto.alert() // 22 anyos
```

Sencillamente, cuando llamas a una función usando el punto, this está implícitamente vinculado al objeto desde el que se llama a la función.

En este ejemplo, dado que alert es llamada desde miObjeto, la palabra clave this está vinculada a miObjeto. Cuando llamamos a alert con miObjeto.alert(), this.edad es 22, siendo edad una propiedad de miObjeto.

Veamos otro ejemplo:

```
function alert() {
  console.log(this.edad + ' anyos');
}
const miObjeto = {
  edad: 22,
  alert: alert,
  anidacionObj: {
    edad: 26,
    alert: alert
  }
}
```

```
miObjeto.anidacionObj.alert(); // 26 anyos
```

Aquí, ya que a alert se la llama desde anidacionObj, this está implícitamente vinculada a anidacionObj en lugar de miObjeto.

Una manera fácil de averiguar a qué objeto está implícitamente vinculado this es mirar qué objeto está a la izquierda del punto (.):

```
function alert() {
  console.log(this.edad + ' anyos');
}

const miObjeto = {
  edad: 22,
  alert: alert,
  anidacionObj: {
    edad: 26,
    alert: alert
  }
}

miObjeto.alert(); // `this` vinculada a `miObjeto` -- 22 anyos
miObjeto.anidacionObj.alert(); // `this` vinculada a `anidacionObj` -- 26 anyos
```

Enlace explícito en JavaScript

Hemos visto que la vinculación implícita tiene que ver con la relación con un objeto.

Pero, ¿qué pasa si queremos **forzar** a una función a usar un objeto como contexto sin poner una referencia de función de propiedad en el objeto?

Tenemos dos métodos para lograr esto: **call() y apply()**.

Junto con un par de otros conjuntos de funciones de utilidad, estas dos utilidades están disponibles para todas las funciones en JavaScript a través del mecanismo [[Prototype]].

Para vincular explícitamente una llamada de función a un contexto, simplemente tienes que invocar la llamada () en esa función y pasar el objeto de contexto como parámetro:

```
function alert() {
  console.log(this.edad + ' anyos');
}

const miObjeto = {
  edad: 22
}

alert.call(miObjeto); // 22 anyos
```

Y aquí está la parte divertida. Incluso si tuvieras que pasar esa función varias veces a nuevas variables (currying), cada invocación usará el mismo contexto porque se ha unido (vinculado explícitamente) a ese objeto. Esto se llama **vinculación dura**.

```
function alert() {
  console.log(this.edad);
}

const miObjeto = {
  edad: 22
};

const bar = function() {
  alert.call(miObjeto);
};
```

```
bar(); // 22
setTimeout(bar, 100); // 22
// una `bar` con vinculación dura ya no puede tener su `this` cambiado
bar.call(window); // sigue siendo 22
```

vinculación dura.

El enlace o vinculación dura es una forma perfecta de unir un contexto en una llamada de función y convertir realmente esa función en un método.