Coerción de tipos

- El demonio de JavaScript -

Chema Durán

Historia de un meme

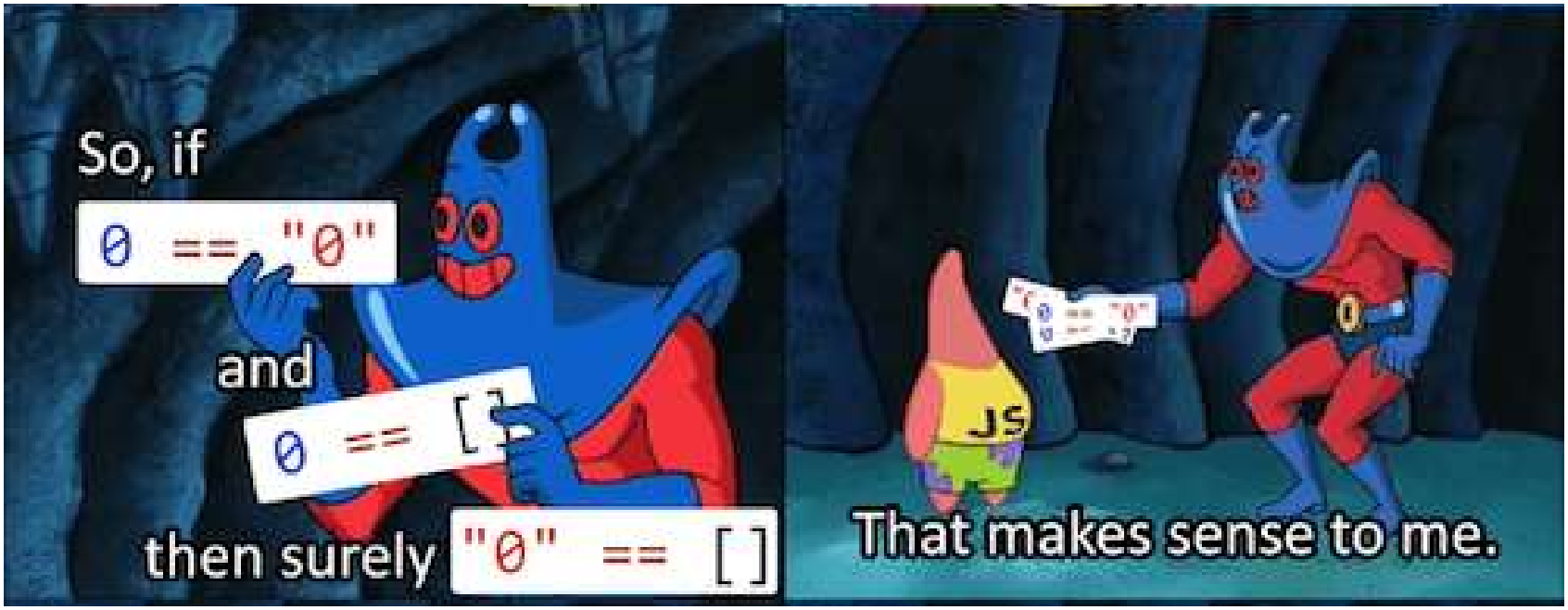
¡Puedes comprobarlo en la consola del desarrollador web en Firefox/Chrome!

Historia de un meme

¡Puedes comprobarlo en la consola del desarrollador web en Firefox/Chrome!

Historia de un meme

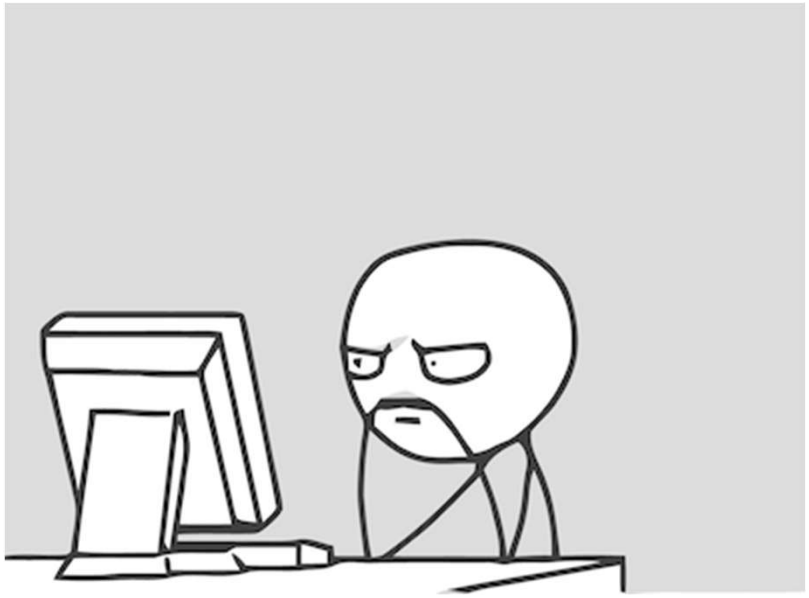
¡Puedes comprobarlo en la consola del desarrollador web en Firefox/Chrome!



Historia de un meme

¡Puedes comprobarlo en la consola del desarrollador web en Firefox/Chrome!

¿Por qué sucede esto?



"I don't think anyone ever really knows JS, not completely anyway."

Kyle Simpson

Parte 1 - Coerción

Si ejecutas `0 == "0"` en la consola, ¿Por qué devuelve `true`?

`0` es un número, y "`0`" es una cadena. ¡Nunca deberían ser lo mismo!. Java (y muchos otros lenguajes) respeta eso. Java devuelve:

```
Error: incomparable types: int and String
```

Eso tiene sentido. Si quieres comparar un `int` y un `String`, primero debes convertirlos a un mismo tipo.

Parte 1 - Coerción

Pero...

Parte 1 - Coerción

Cuando comparas dos valores con el operador ==, uno de los valores puede ir bajo *coerción*.

> **Coerción – convertir automáticamente un valor de un tipo a otro**

***Automáticamente* es la palabra clave aquí. En vez de que tú, explícitamente, conviertas tus tipos, JavaScript lo hace por ti entre bambalinas.**

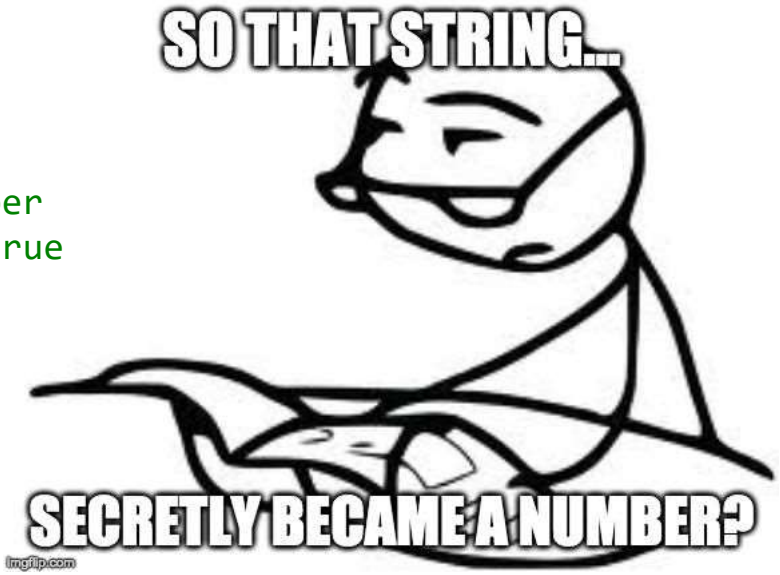
Parte 1 - Coerción

Especificación ECMAScript

Si *a* es Number y *b* es String, devuelve *a* == ToNumber(*b*)

Así para que nuestro caso `0 == "0"`

```
0 == "0"  // true
// El segundo se convierte a Number
// así que...0 es igual a 0, es true
```



Parte 2 –Arrays también “coercionan”

Este sin sentido no está limitado a primitivas como cadenas, números o booleanos. Aquí está nuestra próxima comparación:

```
0 == []  // true
// ¿Qué pasa aquí?
```

¡Coerción otra vez!

Especificación ECMAScript

```
Si a es String o Number y b es Object,
        devuelve a == ToPrimitive(b)
```

Tres cosas pasan aquí:

Parte 2 –Arrays también “coercionan”

1. Sí, los arrays son objects

Lo siento,

la vida es así

Parte 2 –Arrays también “coercionan”

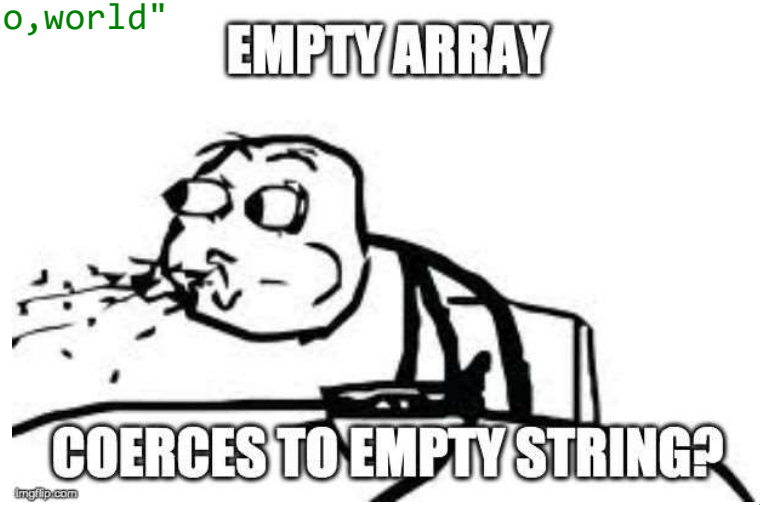
2. Arrays vacíos se convierte a cadena

JavaScript primero mira si el objeto tiene el método toString para hacer la conversión. En el caso de los arrays, existe el método toString que une todos los elementos del array y los devuelve en una cadena.

```
[1, 2, 3].toString() // "1,2,3"
['hello', 'world'].toString() // "hello,world"
```

Y dado que nuestro array está vacío, devolverá una cadena vacía.

```
[].toString() // ""
```



Parte 2 –Arrays también “coercionan”

3. Cadena vacía entonces, se convierte a `0`

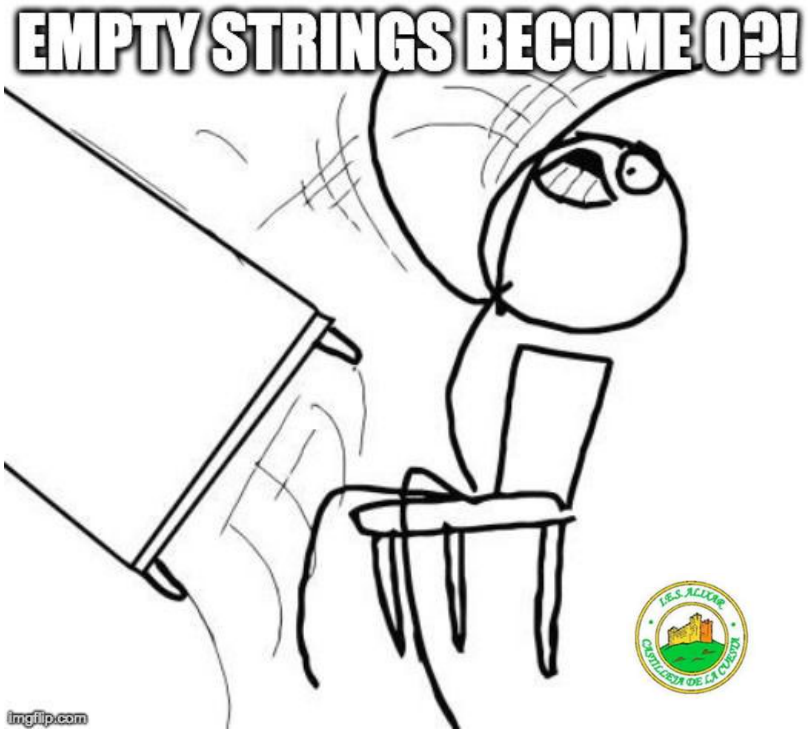
Ya tenemos el array en la cadena vacía.
Volvemos a la primera especificación.

```
Si a es Number y b es String,
devuelve a == ToNumber(b)
```

Así que nuestro `0 == ""`

Como `0` **es** `Number` **y “”** `String`**, devuelve** `0` **== ToNumber(“”)**
ToNumber(“”) devuelve `0`

```
0 == 0
```



Parte 2 –Arrays también “coercionan”

Parte 3 – Resumen

```
0 == "0" // true
```

Porque la coercion convierte esto en `0 == ToNumber("0")`

```
0 == [] // true
```

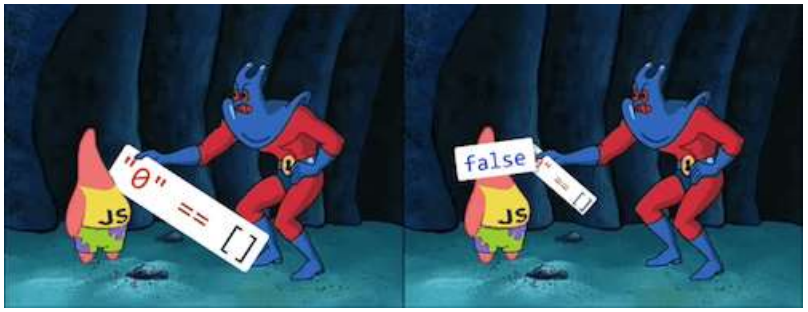
Porque la coercion convierte dos veces, o en dos pasos:

1. `ToPrimitive([])` **nos devuelve un String vacío**
2. **Entonces,** `ToNumber("")` **nos devuelve** `0`

Entonces, de acuerdo a las reglas, ¿Qué debería devolver lo siguiente?

```
"0" == []
```

Parte 4 – false




```
"0" == [] // false
```

Referenciando la especificación una vez más:

```
Si a es String o Number y b es Object, devuelve a == ToPrimitive(b)
```

Eso significa que...

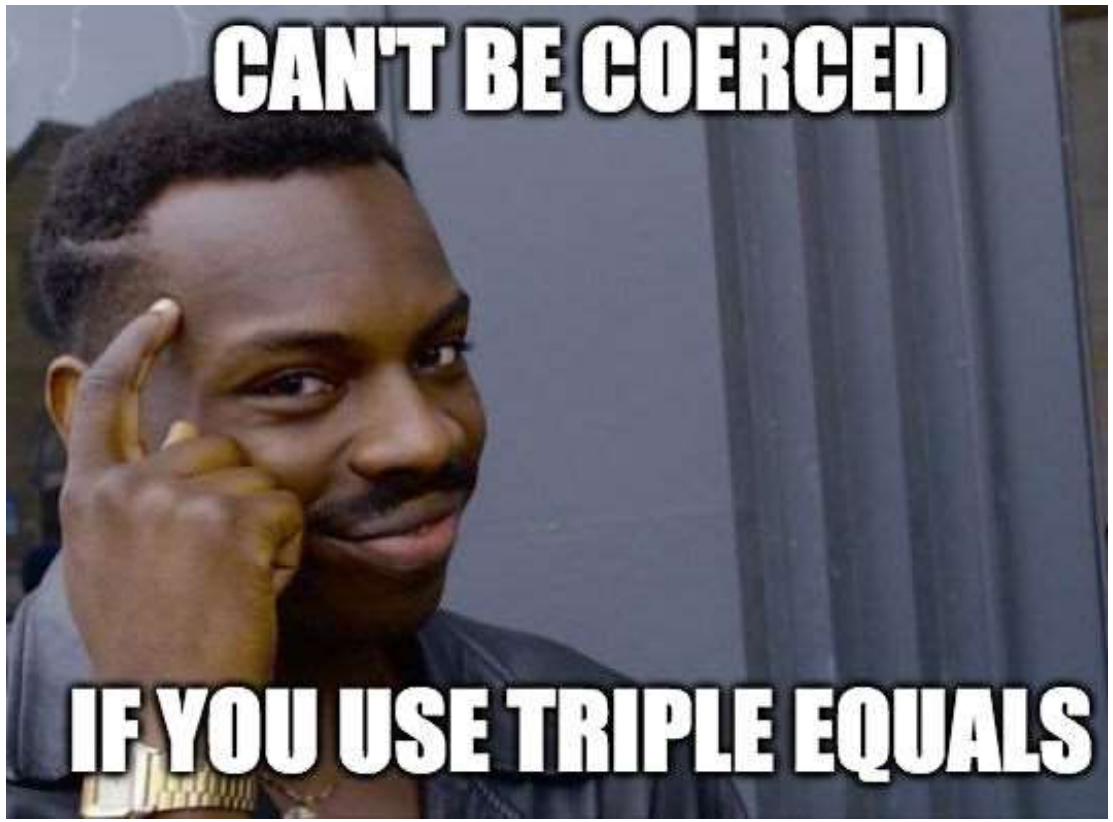
```
Si "0" es String [] es Object, devuelve "0" == ToPrimitive([])
```

`ToPrimitive([])` **nos devuelve un String vacío. La comparación se convierte entonces a:**

```
"0" == ""
```

`"0"` **y** `""` **son Strings, así que JavaScript dice: no necesito más *coerción*. Son disintos, entonces false.**

Conclusión



Usa el operador triple igual (identidad):

```
0 === "0" // false
0 === [] // false
"0" === [] // false
```

Evita por completo la coerción. Por eso mismo además, es más eficiente.

Un solo carácter más, te da mucha más confianza en el código que escribes.

END

prof.jduran@iesalixar.org