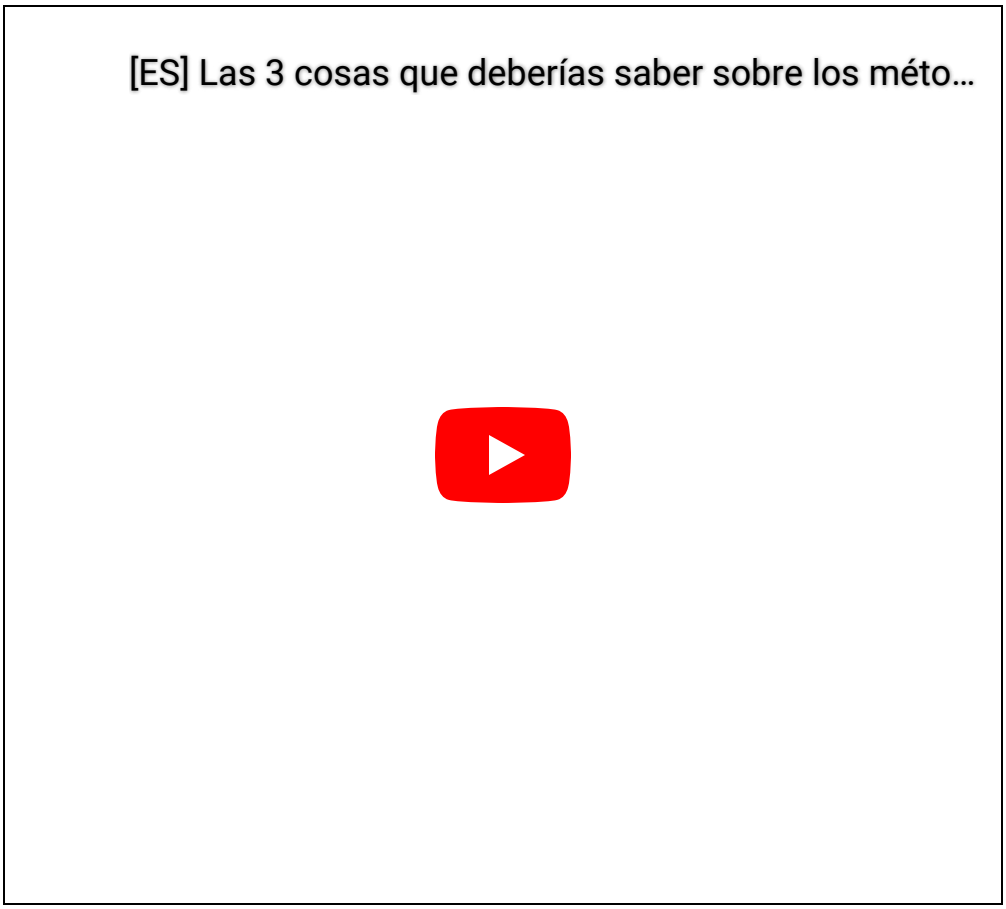
ALBERTO **ALMAGRO**

(https://albertoalmagro.com/)
Freelance Ruby on
Rails Software
Engineer

# Las 3 cosas que deberías saber sobre los métodos bang (!) en Ruby

En el post de esta semana te voy a hablar de un tema que se preguntó en mi canal de YouTube (https://www.youtube.com/channel/UCNDfOyVBiqzRuUjb8wDObZA), que es sobre **cómo utilizar los métodos bang en Ruby**, esos a los que les ponemos una exclamación al final (!). Quédate conmigo para **convertirte en un experto en esto de los métodos bang**. ¡Vamos allá!



# 1. Son una convención

Para Ruby el signo de exclamación, o *bang* (!), no significa nada internamente, **simplemente es una convención de nombrado** que los programadores de Ruby utilizan para denotar que un método puede ser “peligroso”.

De acuerdo con esta convención, el signo de exclamación (!) se debe colocar en el último carácter del nombre del método.

Lo que te recomendaría en estos casos es que **cuando crees un método bang siempre tenga su alternativa sin bang**. En otras palabras, si un método existe con bang, su alternative sin bang debería de existir también.

# 2. Están pensados para llamar tu atención

Los métodos bang se utilizan para indicar peligro o si lo prefieres advertencia, pero, ¿de qué nos advierten realmente? Generalmente será de una de estas dos cosas.

## Este método modifica permanentemente el objeto que recibe el mensaje (efecto secundario)

En las librerías core de Ruby lo normal es que signifique que el método *bang*, a diferencia de su contrapartida sin signo de exclamación, modifique permanentemente al objeto que recibe este mensaje. Por ejemplo, `String#upcase` devuelve un *String* que consiste en el *String* original en mayúsculas, pero sin modificar el String original, mientras que su alternativa bang, `String#upcase!`, sí modifica permanentemente el mismo *String* original convirtiéndolo a mayúsculas.

```
irb(main):001:0> string = "hola"
=> "hola"
irb(main):002:0> string.upcase
=> "HOLA"
irb(main):003:0> string
=> "hola"
irb(main):004:0> string.upcase!
=> "HOLA"
irb(main):005:0> string
=> "HOLA"
```

## Este método lanza una excepción

Este uso es muy típico en Ruby on Rails. Seguro que te has dado cuenta de que en los controladores de Rails se utiliza un patrón diferente cuando se quiere comprobar si una acción como `update` o `save` se ha podido realizar o no. La versión sin bang del método devuelve un valor booleano (`true` si la operación ha tenido éxito y `false` en caso contrario) que normalmente se comprueba en una sentencia `if`, como por ejemplo:

```
def update
  if @post.update(post_params)
    redirect_to @post, notice: "Post was successfully updated."
  else
    render :edit, status: :unprocessable_entity
  end
end
```

Mientras que su contrapartida con bang **asume que la operación tendrá siempre éxito** y lanza una excepción en caso contrario.

```
def update
  @post.update!(post_params) # Raises if couldn't update
end
```

# 3. No son siempre “peligrosos”

¡Ni mucho menos! No son necesariamente peligrosos, de hecho en ocasiones puedes beneficiarte de ellos. Un ejemplo claro es para **ahorrar memoria**. Cuando un método tiene una versión sin bang y otra con bang, la alternativa sin bang crea una nueva instancia en cada ejecución, mientras que la version con bang modifica el objeto que recibe el mensaje, lo que en **colecciones** grandes significa **mayor consumo de memoria** y **más trabajo para el Garbage Collector de Ruby**, lo que a la larga también incrementa el tiempo de ejecución.

Vamos a verlo con un ejemplo comparando `upcase` con `upcase!`. Para ello crearemos un array de strings y ejecutaremos `GC.start` para que el Garbage Collector de Ruby "limpie" antes de la ejecución de nuestro test, a continuación medimos la memoria antes y después de nuestro test para saber la memoria que se ha reservado:

## upcase

```ruby
data = Array.new(1000) { 'x' * 1024 * 1024 }
GC.start
memory_before = %x[ps -o rss= -p #{Process.pid}].to_i/1024
data.map(&:upcase)
memory_after = %x[ps -o rss= -p #{Process.pid}].to_i/1024

puts 'upcase: %d MB' % (memory_after - memory_before)
```

Con `upcase` el resultado fue el siguiente

```
2.7.3 $ ruby performance.rb
upcase: 1005 MB
```

## upcase!

```ruby
data = Array.new(1000) { 'x' * 1024 * 1024 }
GC.start
memory_before = %x[ps -o rss= -p #{Process.pid}].to_i/1024
data.map(&:upcase!)
memory_after = %x[ps -o rss= -p #{Process.pid}].to_i/1024

puts 'upcase!: %d MB' % (memory_after - memory_before)
```

Y con `upcase!` el resultado fue:

```
2.7.3 $ ruby performance_bang.rb
upcase: 0 MB
```

Como puedes ver la versión sin *bang* reserva un extra de 1005MB mientras que l**a versión con *bang* no necesita ningún extra, así que cuando vayas a ejecutar muchas operaciones sobre una colección ten esto en cuenta** 😉

¡Y esto es todo lo que quería comentarte sobre los métodos bang en Ruby! Espero que te haya gustado y te haya servido para aprender o refrescar conceptos. Recuerda suscribirte para no perderte ninguna de mis actualizaciones y si tienes alguna duda no dejes pasar la oportunidad y pregúntame en la sección de comentarios.

Nos vemos la semana que viene, ¡adiós!

ALBERTO ALMAGRO

(https://albertoalmagro.com/)

Freelance Ruby on Rails Software Engineer

Legal Notice (https://albertoalmagro.com/legal-notice/)

Cookies policy (https://albertoalmagro.com/cookies-policy/)

Privacy policy (https://albertoalmagro.com/privacy-policy/)