

# Programación 00

*Julio 2019*

Traducido por Leonardo Collado-Torres, con modificaciones de Alejandro Reyes

@fellgernon @areyesq

lcolladotor@gmail.com alejandro.reyes.ds@gmail.com

lcolladotor.github.io alejandroreyes.org/

Desarrollado por Charlotte Wickham para rstudio::conf(2019)

@cvwickham

cwickham@gmail.com

cwick.co.nz

Adapted from *Tidy Tools* by Hadley Wickham



# ¿Qué es S3?

¿Qué es lo que hace **S3**?

# S3 potencia el comportamiento contexto-específico

```
x <- 1:5  
y <- factor(letters[1:5])
```

```
summary(x)
```

#	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
#	1	2	3	3	4	5

```
summary(y)
```

#	a	b	c	d	e
#	1	1	1	1	1

Un resumen de 6 números

Una tabla de categorías

# summary() es un S3 generic

```
sloop::ftype(summary)
```

```
# [1] "S3"           "generic"
```

```
# summary() busca métodos basados en la  
# clase de los objetos
```

```
sloop::s3_class(y)
```

```
# [1] "factor"
```

```
sloop::s3_dispatch(summary(y))
```

```
# => summary.factor => llama a este método
```

```
# * summary.default* este método existe pero no fue llamado
```

# Tu turno

```
mod <- lm(mpg ~ wt, data = mtcars)  
summary(mod)
```

¿Cuál es la **clase** de mod?

¿Qué **método** fue utilizado (*dispatched*) por  
summary()?

¿Puedes encontrar el código del método utilizado?

```
library(sloop)
mod <- lm(mpg ~ wt, data = mtcars)
summary(mod)

s3_class(mod)
# [1] "lm"

s3_dispatch(summary(mod))
# => summary.lm
# * summary.default

summary.lm

# no siempre va a funcionar
# usa `s3_get_method()` para encontrar métodos no exportados
s3_get_method(summary.lm)
```

# Motivación

¿Por qué te debería importar S3?



Ya estás usando objetos S3 en tus  
análisis

# Objetos S3 importantes en R base

`data.frame()`

`factor()`

`Sys.Date()`

`Sys.time()`

`table()`



Funciones complejas tienen que  
regresar varias cosas

Esto es claramente importante en modelos lineales

```
mod <- lm(mpg ~ wt, data = mtcars)  
str(mod)
```

# Pero también en sus resúmenes

```
sum <- summary(mod)  
str(sum)
```

A close-up, low-angle shot of the iconic undulating facade of the Walt Disney Concert Hall. The facade is composed of numerous thin, light-colored metal panels that curve and overlap, creating a dynamic, wave-like pattern. The lighting highlights the metallic texture and the shadows between the panels.

La forma sigue  
a la función

# Un ejemplo son modelos lineales

```
sum
#> Call:
#> lm(formula = mpg ~ wt, data = mtcars)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -4.5432 -2.3647 -0.1252  1.4096  6.8727
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 37.2851    1.8776 19.858 < 2e-16 ***
#> wt          -5.3445    0.5591 -9.559 1.29e-10 ***
#> ---
#> Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
#>
#> Residual standard error: 3.046 on 30 degrees of freedom
#> Multiple R-squared:  0.7528, Adjusted R-squared:  0.7446
#> F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

# Otro ejemplo son tibbles

Tamaño total

```
# A tibble: 53,940 x 10
```

	carat	cut	color	clarity	depth	table	price	x	y	z
1	<dbl>	<ord>	<ord>	<ord>	<dbl>	<dbl>	<int>	<dbl>	<dbl>	<dbl>
2	0.230	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
3	0.210	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
4	0.230	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
5	0.290	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
6	0.310	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
7	0.240	"Very Good"	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
8	0.240	"Very Good"	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
9	0.260	"Very Good"	H	SI1	61.9	55.0	337	4.07	4.11	2.53
10	0.220	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49

```
# ... with 53,930 more rows
```

Tipo de variable

Solo muestra las primeras 10 líneas

S3 hace que los paquetes sean posibles de extender

## Nuevos métodos

Te permite extender otros paquetes

## Nuevos genéricos

Escribe paquetes de tal forma  
que otros los puedan extender

# Clases “escalares”

un solo objeto complejo

# Principio:

Provee una estructura  
consistente y un método de  
 impresión para resultados  
 complejos

Cambia al proyecto:  
[safely]

# Reto: ¿cómo podría mejorar el resultado de safely?

```
library(purrr)
safe_log <- safely(log)

safe_log("a")
#> $result
#> NULL
#>
#> $error
#> <simpleError in log(...):
#>   non-numeric argument to
#>   mathematical function>

safe_log(10)
#> $result
#> [1] 2.302585
#>
#> $error
#> NULL
```

# Crea una nueva clase S3

1. Determina un nombre **safely**
2. Define las propiedades de la clase
3. Escribe el constructor
4. Escribe métodos

# Tu turno

¿Cuáles son los invariantes del resultado de `safely`?

```
safe_log <- purrr::safely(log)
```

```
safe_log(x)
```

```
# ¿Qué sabemos que siempre es cierto  
# del resultado de safe_log(x)?
```

# Invariantes

Regresa una list

- dos componentes: resultado y error
- el resultado siempre debería venir primero
- uno siempre es NULL

# Ahora, escribe el constructor

```
new_safely <- function(result = NULL, error = NULL) {  
  if (!is.null(result) && !is.null(error)) {  
    stop(  
      "One of `result` and `error` must be NULL",  
      call. = FALSE  
    )  
  }  
}
```

Revisa los valores de entrada

```
structure(  
  list(  
    result = result,  
    error = error  
  class = "safely"  
)  
}
```

Establece la estructura y determina la clase

# Definición de safely

```
safely <- function(.f) {  
  stopifnot(is.function(.f))  
  
  function(...) {  
    tryCatch({  
      list(result = .f(...), error = NULL)  
    }, error = function(e) {  
      list(result = NULL, error = e)  
    })  
  }  
}
```

# Ahora usa el constructor

```
safely <- function(.f) {  
  stopifnot(is.function(.f))  
  
  function(...) {  
    tryCatch({  
      new_safely(result = .f(...))  
    }, error = function(e) {  
      new_safely(error = e)  
    })  
  }  
}
```

# Abreviación

# Prueba

`expect_null()`

Checa si es un NULL literal

`expect_type()`  
`expect_s3_class()`  
`expect_s4_class()`

Checa que herede de un tipo base,  
clase S3, o clase S4.

`expect_true()`  
`expect_false()`

Captura las expectaciones para los  
escenarios no cubiertos por otras  
funciones

# Tu turno

Escribe pruebas para asegurar que nuestra nueva función `new_safely()` regrese el valor de retorno esperado sin importar que haya ocurrido un error. (es decir, expresa los invariantes como unidades de prueba)

```
# En tests/testthat/test-safely.R
context("test-safely.R")

test_that("solo podemos prover un error o resultado", {
  expect_error(new_safely(1, 2), "must be NULL")
})

test_that("está bien que ambos sean nulos", {
  expect_error(new_safely(NULL, NULL), NA)
})

test_that("resultado y error son capturados", {
  s1 <- new_safely(result = 1)
  s2 <- new_safely(error = 1)

  expect_s3_class(s1, "safely")
  expect_equal(s1$result, 1)
  expect_equal(s1$error, NULL)

  expect_s3_class(s2, "safely")
  expect_equal(s2$result, NULL)
  expect_equal(s2$error, 1)
})
```

Espera que no haya error

Ahora podemos mejorar el resultado con un método para imprimir

```
safe_log(10)
#> <safely: ok>
#> [1] 2.302585
```

Creo que es buena  
práctica incluir el tipo en



```
safe_log("a")
#> <safely: error>
#> Error: non-numeric argument to
#> mathematical function
```

# Todos los métodos S3 tienen la misma estructura básica

genérico

```
print.safely <- function(x, ...) {
```

clase

```
}
```

Los mismos argumentos que el genérico

Métodos  
perteneцен a  
genéricos, no a  
clases

`print`

`mean`

`sum`

Date

POSIXct integer

Date

POSIXct integer

print

mean

sum

# Tu turno: llena los espacios vacíos

```
# En R/safely.R
print.safely <- function(x, ...) {
}

# Una función de ayuda útil que está en utils.R
cat_line <- function(...) {
  cat(..., "\n", sep = "")
}
# Checa https://github.com/r-lib/cli para
# más funciones de ayuda.
```

# Algunos casos de prueba

```
f <- function() stop("mensaje")
```

```
g <- function() 1
```

```
safe_f <- safely(f)
```

```
safe_g <- safely(g)
```

```
safe_f()
```

```
safe_g()
```

# Mi método de imprimir (**print**)

```
print.safely <- function(x, ...) {  
  if (!is.null(x$error)) {  
    cat_line("<safely: error>")  
    cat_line("Error: ", x$error$message)  
  } else {  
    cat_line("<safely: ok>")  
    print(x$result)  
  }  
}
```

```
invisible(x)  
}
```

Usado principalmente por  
los efectos secundarios

# Un poco de color puede ser transformative

```
print.safely <- function(x, ...) {  
  if (!is.null(x$error)) {  
    cat_line("<safely: ", crayon::bold(crayon::red("error")), ">")  
    cat_line(crayon::red("Error: "), x$error$message)  
  } else {  
    cat_line("<safely: ", crayon::green("ok"), ">")  
    print(x$result)  
  }  
  
  invisible(x)  
}
```

# Nuevo genérico

Cambia al proyecto:  
[bizarro]

# Objetivo: crear la función bizarro

```
bizarro("abc")
```

```
#> [1] "cba"
```

```
bizarro(1)
```

```
#> [1] -1
```

```
bizarro(c(TRUE, FALSE))
```

```
#> [1] FALSE TRUE
```

# Podríamos usar if + else

```
str_reverse <- function(x) {  
  purrr::map_chr(stringr::str_split(x, ""),  
    ~ stringr::str_flatten(rev(.x)))  
}  
  
bizarro <- function(x) {  
  if (is.character(x)) {  
    str_reverse(x)  
  } else if (is.numeric(x)) {  
    -x  
  } else if (is.logical(x)) {  
    !x  
  } else {  
    stop(  
      "Don't know how to make bizzaro <", class(x)[[1]], ">",  
      call. = FALSE)  
  }  
}
```

# Pero en vez vamos a crear nuestro genérico S3

```
bizarro <- function(x) {  
  UseMethod("bizarro")  
}
```

Mágicamente pasa los argumentos al lugar correcto

genérico.clase

```
bizarro.character <- function(x) {  
  str_reverse(x)  
}
```

```
bizarro("abc")  
#> [1] cba
```

Le permite a cualquiera extender

# Tu turno

Implementa:

1. un método numérico (**numeric**) que multiplique por -1
2. un método lógico (**logical**) que invierta TRUE/FALSE
3. un método **data frame** que corra buzzard en los nombres de las columnas, al igual que en cada columna.

(es decir, logra que pasen las unidades de prueba)

```
bizarro.numeric <- function(x) {  
  -x  
}
```

```
bizarro.logical <- function(x) {  
  !x  
}
```

```
bizarro.data.frame <- function(x) {  
  names(x) <- bizarro(names(x))  
  x[] <- purrr::map(x, bizarro)  
  x  
}
```

**Técnica útil** Método para objeto complejo: aplica el genérico a los componentes.

# ¿Qué pasa si un método no está disponible?

```
bizarro(factor(letters))
#> Error in UseMethod("bizarro") :
#>   no applicable method for 'bizarro'
#>   applied to an object of class "factor"

# ¿Podemos hacer algo mejor?
# Necesitamos proveer un método de defecto que
# funcione en estos casos
```

```
bizarro.default <- function(x) {  
  stop(  
    "Don't know how to make bizzaro < ",  
    class(x)[[1]], ">",  
    call. = FALSE  
)  
}
```

```
bizarro(factor(letters))  
#> Error: Don't know how to make  
#> bizzaro <factor>
```

# Tu turno

¿Qué debería regresar bizzaro.factor("abc"))?

Decide, codifica tus decisiones en unidades de prueba, y luego implementa bizarro.factor().

# Una idea: invertir las letras en los niveles del factor

```
# En tests/testthat/test-bizarro.R
test_that("bizarro factors have levels reversed", {
  f1 <- factor(c("abc", "def", "abc"))
  f2 <- factor(c("cba", "fed", "cba"))

  expect_equal(bizarro(f1), f2)
  expect_equal(bizarro(f2), f1)
})
```

```
# En R/bizarro.R  
bizarro.factor <- function(x) {  
  levels(x) <- bizarro(levels(x))  
  x  
}
```

Para aprender  
más

Advanced R (2nda ed) tiene cuatro capítulos

**S3:** <https://adv-r.hadley.nz/s3.html>

**S4:** <https://adv-r.hadley.nz/s4.html>

**R6:** <https://adv-r.hadley.nz/r6.html>

**Concesiones:** <https://adv-r.hadley.nz/oo-tradeoffs.html>

# Clases de vectores

Clases construidas en otros tipos de vectores



"vctrs típicamente será usado por otros paquetes, facilitando el que puedan proveer nuevas clases de vectores S3 que estén soportados en todo el tidyverse (y más allá)."

--<https://vctrs.r-lib.org/>



Adapted from *Tidy Tools* by Hadley Wickham

This work is licensed as  
Creative Commons  
Attribution-ShareAlike 4.0  
International

To view a copy of this license, visit  
[https://creativecommons.org/  
licenses/by-sa/4.0/](https://creativecommons.org/licenses/by-sa/4.0/)