EXPLOSIVE DEBUGGING WITH BOOMER!

boomer

Antoine Fabri

cynkra

Debugging?

- Debugging = fixing the logic

Debugging?

- Debugging = fixing the logic
  - From the code itself

Debugging?

- Debugging = fixing the logic
  - From the code itself
  - By checking whether values match expectations
    - Interactively
      - `browser()`, `debug()`, `debugonce()`
    - By modifying the code to print values
      - To the console
      - To log files

Added value

- *boomer* makes things simpler
  - No need to modify the code
  - No need to type print 100 times
- *boomer* "explodes" the calls and prints all intermediate values


50:50
WHO WANTS TO BE A
MILLIONAIRE
WHO WANTS TO BE A
!TRUE || TRUE == FALSE && FALSE
A: TRUE
B: FALSE
C: NA
D: object of type 'closure' is n...

boom()

```
> res <- boom(!TRUE || TRUE == FALSE && FALSE)
💣 !TRUE || TRUE == FALSE && FALSE
· 💣 💥 !TRUE
· [1] FALSE
·
· 💣 TRUE == FALSE && FALSE
· · 💣 💥 TRUE == FALSE
· · [1] FALSE
· ·
· 💥 TRUE == FALSE && FALSE
· [1] FALSE
·
💥 !TRUE || TRUE == FALSE && FALSE
[1] FALSE
```

boom()

```
> res <- boom(subset(cars[1:2, ], dist == 10))
💣 subset(cars[1:2, ], dist == 10)
· 💣 cars[1:2, ]
· · 💣 💥 1:2
· · [1] 1 2
· ·
· 💥 cars[1:2, ]
·   speed dist
· 1     4    2
· 2     4   10
·
· 💣 💥 dist == 10
· [1] FALSE  TRUE
·
💥 subset(cars[1:2, ], dist == 10)
  speed dist
2     4   10
```

cynkra.

boom()

- `constructive::construct()` prints the code to rebuild the objects
- Can be used for more precision

```
> options(boomer.print = constructive::construct)
> x <- boom(subset(cars[1:2, ], dist == 10))
💣 subset(cars[1:2, ], dist == 10)
· 💣 cars[1:2, ]
· · 💣 💥 1:2
· · 1:2
· ·
· 💥 cars[1:2, ]
· data.frame(speed = 4, dist = c(2, 10))
·
· 💣 💥 dist == 10
· c(FALSE, TRUE)
·
💥 subset(cars[1:2, ], dist == 10)
data.frame(speed = 4, dist = 10, row.names = 2L)
```

cynkra.

boom()

We can use *boomer* to measure performance too!

```
> options(boomer.print = class, boomer.clock = TRUE)
> x <- boom(subset(cars[1:2, ], dist == 10))
💣 subset(cars[1:2, ], dist == 10)
· 💣 cars[1:2, ]
· · 💣 💥 1:2
time: 0.007 ms
· · [1] "integer"
· ·
· 💥 cars[1:2, ]
time: 0.353 ms
· [1] "data.frame"
·
· 💣 💥 dist == 10
time: 0.01 ms
· [1] "logical"
·
💥 subset(cars[1:2, ], dist == 10)
time: 0.004 s
[1] "data.frame"
```

cynkra.

rig()

```
> sd_rigged <- rig(sd)
> res <- sd_rigged(c(1, 2, 3))
👆 stats::sd
💣 sqrt(var(if (is.vector(x) || is.factor(x)) x else as.double(x),...
· 💣 var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm = na.rm)
· · 💣 if (is.vector(x) || is.factor(x)) x else as.double(x)
· · · 💣 is.vector(x) || is.factor(x)
· · · · x :
· · · · [1] 1 2 3
· · · · 💣 💥 is.vector(x)
· · · · [1] TRUE
· · · ·
· · · 💥 is.vector(x) || is.factor(x)
· · · [1] TRUE
· · ·
· · 💥 if (is.vector(x) || is.factor(x)) x else as.double(x)
· · [1] 1 2 3
· ·
· 💥 var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm = na.rm)
· [1] 1
·
💥 sqrt(var(if (is.vector(x) || is.factor(x)) x else as.double(x),
    na.rm = na.rm
  ))
[1] 1

👆 stats::sd
```

```
> sd
function (x, na.rm = FALSE)
sqrt(var(if (is.vector(x) || is.factor(x)) x else as.double(x),
    na.rm = na.rm))
<bytecode: 0x106d87c68>
<environment: namespace:stats>
```

cynkra.

rig_in_place()

```
> rig_in_place(sd)
> sd(c(1, 2, 3))
👆 sd
💣 sqrt(var(if (is.vector(x) || is.factor(x)) x else as.double(x),...
· 💣 var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm = na.rm)
· · 💣 if (is.vector(x) || is.factor(x)) x else as.double(x)
· · · 💣 is.vector(x) || is.factor(x)
· · · · x :
· · · · [1] 1 2 3
· · · · 💣 💥 is.vector(x)
· · · · [1] TRUE
· · · ·
· · · 💥 is.vector(x) || is.factor(x)
· · · [1] TRUE
· · ·
· · 💥 if (is.vector(x) || is.factor(x)) x else as.double(x)
· · [1] 1 2 3
· ·
· 💥 var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm = na.rm)
· [1] 1
·
💥 sqrt(var(if (is.vector(x) || is.factor(x)) x else as.double(x),
     na.rm = na.rm
   ))
[1] 1

👆 sd
[1] 1
```

cynkra.

boom_on() / boom_off()

```
> rle(c("a", "a", "b", "b", "b"))
debugging in: rle(c("a", "a", "b", "b", "b"))
Browse[1]>
```

```
rle <- function (x)
{
    if (!is.vector(x) && !is.list(x))
        stop("'x' must be a vector of an atomic type")
    n <- length(x)
    if (n == 0L)
        return(structure(list(lengths = integer(), values = x),
            class = "rle"))
    y <- x[-1L] != x[-n]
    i <- c(which(y | is.na(y)), n)
    structure(list(lengths = diff(c(0L, i)), values = x[i]),
        class = "rle")
}
```

boom_on() / boom_off()

```
> rle(c("a", "a", "b", "b", "b"))
debugging in: rle(c("a", "a", "b", "b", "b"))
Browse[1]>
Browse[1]>
```

```
rle <- function (x)
{
    if (!is.vector(x) && !is.list(x))
        stop("'x' must be a vector of an atomic type")
    n <- length(x)
    if (n == 0L)
        return(structure(list(lengths = integer(), values = x),
            class = "rle"))
    y <- x[-1L] != x[-n]
    i <- c(which(y | is.na(y)), n)
    structure(list(lengths = diff(c(0L, i)), values = x[i]),
        class = "rle")
}
```

boom_on() / boom_off()

```
> rle(c("a", "a", "b", "b", "b"))
debugging in: rle(c("a", "a", "b", "b", "b"))
Browse[1]>
Browse[1]>
Browse[1]>
```

```
rle <- function (x)
{
    if (!is.vector(x) && !is.list(x))
        stop("'x' must be a vector of an atomic type")
    n <- length(x)
    if (n == 0L)
        return(structure(list(lengths = integer(), values = x),
            class = "rle"))
    y <- x[-1L] != x[-n]
    i <- c(which(y | is.na(y)), n)
    structure(list(lengths = diff(c(0L, i)), values = x[i]),
        class = "rle")
}
```

boom_on() / boom_off()

```
> rle(c("a", "a", "b", "b", "b"))
debugging in: rle(c("a", "a", "b", "b", "b"))
Browse[1]>
Browse[1]>
Browse[1]>
Browse[1]>
```

```
rle <- function (x)
{
    if (!is.vector(x) && !is.list(x))
        stop("'x' must be a vector of an atomic type")
    n <- length(x)
    if (n == 0L)
        return(structure(list(lengths = integer(), values = x),
            class = "rle"))
    y <- x[-1L] != x[-n]
    i <- c(which(y | is.na(y)), n)
    structure(list(lengths = diff(c(0L, i)), values = x[i]),
        class = "rle")
}
```

boom_on() / boom_off()

```
> rle(c("a", "a", "b", "b", "b"))
debugging in: rle(c("a", "a", "b", "b", "b"))
Browse[1]>
Browse[1]>
Browse[1]>
Browse[1]>
Browse[1]>
```

```
rle <- function (x)
{
    if (!is.vector(x) && !is.list(x))
        stop("'x' must be a vector of an atomic type")
    n <- length(x)
    if (n == 0L)
        return(structure(list(lengths = integer(), values = x),
            class = "rle"))
    y <- x[-1L] != x[-n]
    i <- c(which(y | is.na(y)), n)
    structure(list(lengths = diff(c(0L, i)), values = x[i]),
        class = "rle")
}
```

boom_on() / boom_off()

```
> rle(c("a", "a", "b", "b", "b"))
debugging in: rle(c("a", "a", "b", "b", "b"))
Browse[1]>
Browse[1]>
Browse[1]>
Browse[1]>
Browse[1]> boom_on()
Browse[1]>
```

```
rle <- function (x)
{
    if (!is.vector(x) && !is.list(x))
        stop("'x' must be a vector of an atomic type")
    n <- length(x)
    if (n == 0L)
        return(structure(list(lengths = integer(), values = x),
            class = "rle"))
    y <- x[-1L] != x[-n]
    i <- c(which(y | is.na(y)), n)
    structure(list(lengths = diff(c(0L, i)), values = x[i]),
        class = "rle")
}
```

boom_on() / boom_off()

```
Browse[1]> boom_on()
Browse[1]>
💣 x[-1L] != x[-n]
· 💣 x[-1L]
· · 💣 💥 -1L
· · [1] -1
· ·
· 💥 x[-1L]
· [1] "a" "b" "b" "b"
·
· 💣 x[-n]
· · 💣 💥 -n
· · [1] -5
· ·
· 💥 x[-n]
· [1] "a" "a" "b" "b"
·
💥 x[-1L] != x[-n]
[1] FALSE  TRUE FALSE FALSE

[1] FALSE  TRUE FALSE FALSE
Browse[1]>
```

```
rle <- function (x)
{
    if (!is.vector(x) && !is.list(x))
        stop("'x' must be a vector of an atomic type")
    n <- length(x)
    if (n == 0L)
        return(structure(list(lengths = integer(), values = x),
            class = "rle"))
    y <- x[-1L] != x[-n]
    i <- c(which(y | is.na(y)), n)
    structure(list(lengths = diff(c(0L, i)), values = x[i]),
        class = "rle")
}
```

cynkra.

boom_on() / boom_off()

```
Browse[1]>
💣 c(which(y | is.na(y)), n)
· 💣 which(y | is.na(y))
· · 💣 y | is.na(y)
· · · 💣 💥 is.na(y)
· · · [1] FALSE FALSE FALSE FALSE
· · ·
· · 💥 y | is.na(y)
· · [1] FALSE  TRUE FALSE FALSE
· ·
· 💥 which(y | is.na(y))
· [1] 2
·
💥 c(which(y | is.na(y)), n)
[1] 2 5

[1] 2 5
Browse[1]>
```

```
rle <- function (x)
{
    if (!is.vector(x) && !is.list(x))
        stop("'x' must be a vector of an atomic type")
    n <- length(x)
    if (n == 0L)
        return(structure(list(lengths = integer(), values = x),
            class = "rle"))
    y <- x[-1L] != x[-n]
    i <- c(which(y | is.na(y)), n)
    structure(list(lengths = diff(c(0L, i)), values = x[i]),
        class = "rle")
}
```

boom_on() / boom_off()

```
Browse[1]>
💣 c(which(y | is.na(y)), n)
· 💣 which(y | is.na(y))
· · 💣 y | is.na(y)
· · · 💣 💥 is.na(y)
· · · [1] FALSE FALSE FALSE FALSE
· · ·
· · 💥 y | is.na(y)
· · [1] FALSE  TRUE FALSE FALSE
· ·
· 💥 which(y | is.na(y))
· [1] 2
·
💥 c(which(y | is.na(y)), n)
[1] 2 5

[1] 2 5
Browse[1]> boom_off()
Browse[1]>
```

```
rle <- function (x)
{
    if (!is.vector(x) && !is.list(x))
        stop("'x' must be a vector of an atomic type")
    n <- length(x)
    if (n == 0L)
        return(structure(list(lengths = integer(), values = x),
            class = "rle"))
    y <- x[-1L] != x[-n]
    i <- c(which(y | is.na(y)), n)
    structure(list(lengths = diff(c(0L, i)), values = x[i]),
        class = "rle")
}
```

cynkra.

How does it work?

Same code!

```
> identical(body(sd_rigged), body(sd))
[1] TRUE
```

How does it work?

Different environment!

When we call `sd()`:

- The parent of the execution environment is the "stats" namespace

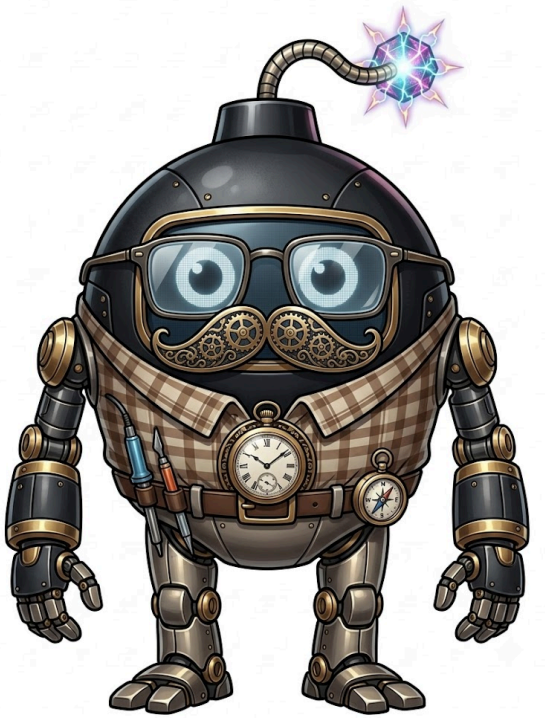
When we call `sd_rigged()`:

- The parent of the execution environment is a mask full of shims
- This mask's parent is the "stats" namespace

```
> parent.env(environment(sd_rigged))
<environment: namespace:stats>
> names(environment(sd_rigged))
 [1] ":::"       "::"        "as.double" "var"       "="         "sqrt"
 [7] "("         "if"        "is.factor" "<-"        "||"        "is.vector"
```

Give logs to your AI

```r
# in your package (during development)
.onLoad <- function(libname, pkgname) {
    options(
    # log to a file rather than to the console,
    boomer.log = "boomer_execution_log.log",
    # faithful representation of objects,
    boomer.print = constructive::construct,
    # measure the time of each step
    boomer.clock = TRUE,
    # log more context, good for AI
    boomer.ignore = character()
  )
  # rig everything at the next devtools::load_all()
  boomer::rig_in_namespace(fun1, fun2)
}
```



Thank you!

cynkra.

github.com/moodymudskipper/boomer · antoine@cynkra.com

Back on CRAN soon!