Advanced R

Chapter 10

R4DS Reading Group

Overview

- What is a function factory?
    - Function factories and manufactured functions
    - Manufactured function environments
    - Promises and `force`
    - {factory}
- Why use a function factory?
    - Stateful functions
    - ggplot2
    - Expensive calculations

```r
library(rlang)
library(ggplot2)
library(scales)
# remotes::install_github("jonthegeek/factory")
library(factory)
```

What is a function factory?

Function factories and manufactured functions

```r
# Function factory
power1 <- function(exponent) {
  function(x) {
    x ^ exponent
  }
}

# Manufactured functions
square1 <- power1(2)
cube1 <- power1(3)
square1(8)
```

```
## [1] 64
```

Manufactured function environments

```
square1
```

```
## function(x) {
##     x ^ exponent
##   }
## <environment: 0x00000079cb22eed0>
```

```
cube1
```

```
## function(x) {
##     x ^ exponent
##   }
## <bytecode: 0x00000079caf789c0>
## <environment: 0x00000079b315fc08>
```

```
c(fn_env(square1)$exponent, fn_env(cube1)$exponent)
```

```
## [1] 2 3
```

Promise dangers

Lazy evaluation + factories = danger

```r
my_exponent <- 2
square1b <- power1(my_exponent)
my_exponent <- 3
square1b(2)
```

```
## [1] 8
```

Forcing evaluation

`force` forces evaluation

```r
power2 <- function(exponent) {
  force(exponent)
  function(x) {
    x ^ exponent
  }
}
```

(technically just `exponent` instead of `force(exponent)` does the same thing)

{factory}

- I created a package to handle some of the fancy stuff.
- Maybe discuss internals in Chapter 19?

```r
power3 <- factory::build_factory(
  function(x) {
    x ^ exponent
  },
  exponent
)
my_exponent <- 2
square3 <- power3(my_exponent)
my_exponent <- 3
square3(2)
```

```
## [1] 4
```

```r
square3
```

```
## function (x)
## {
##     x^2
## }
```

Why use a function factory?

Stateful functions

```r
new_guessing_game <- function() {
  target <- sample(1:100, 1)
  previous_diff <- NA_integer_
  function(guess) {
    if (guess %in% 1:100) {
      if (guess == target) {
        message("Correct!")
        return(invisible(TRUE))
      }
      new_diff <- abs(target - guess)
      if (is.na(previous_diff) || new_diff == previous_diff) {
        message("Try again!")
      } else if (new_diff < previous_diff) message("Warmer!")
      else message("Colder!")
      previous_diff <<- new_diff
    } else stop("Your guess should be between 1 and 100.")
    return(invisible(FALSE))
  }
}
```

Stateful functions (cont)

```
guess <- new_guessing_game()
guess(50)
```

```
## Try again!
```

```
guess(75)
```

```
## Colder!
```

```
guess(50)
```

```
## Warmer!
```

```
guess(25)
```

```
## Warmer!
```

```
guess(50)
```

```
## Colder!
```

{ggplot2}

Lots of ggplot2 functions accept functions as arguments

```
?ggplot2::geom_histogram
```

> `binwidth` The width of the bins. Can be specified as a numeric value or as a function that calculates width from unscaled x. Here, "unscaled x" refers to the original x values in the data, before application of any scale transformation. When specifying a function along with a grouping structure, the function will be called once per group...

{scales}

The {scales} package is full of function factories.

```
scales::number_format
```

```
## function (accuracy = NULL, scale = 1, prefix = "", suffix = "",
##     big.mark = " ", decimal.mark = ".", trim = TRUE, ...)
## {
##     force_all(accuracy, scale, prefix, suffix, big.mark, decimal.mark,
##         trim, ...)
##     function(x) number(x, accuracy = accuracy, scale = scale,
##         prefix = prefix, suffix = suffix, big.mark = big.mark,
##         decimal.mark = decimal.mark, trim = trim, ...)
## }
## <bytecode: 0x00000079caab4380>
## <environment: namespace:scales>
```

Expensive calculations

```r
boot_model <- function(df, formula) {
  # Pretend these calculations would be slow
  mod <- lm(formula, data = df)
  fitted_vals <- unname(fitted(mod))
  resid_vals <- unname(resid(mod))
  rm(mod) # Or use {factory} and this won't be necessary!
  function() {
    fitted_vals + sample(resid_vals)
  }
}
boot_mtcars1 <- boot_model(mtcars, mpg ~ wt)
head(boot_mtcars1())
```

```
## [1] 19.55574 22.78664 20.98059 17.82004 17.80000 21.14321
```

```r
head(boot_mtcars1())
```

```
## [1] 22.36284 19.83382 25.75283 20.45908 23.06388 18.59311
```

Expensive calculations (cont)

```r
boot_mtcars1
```

```
## function() {
##     fitted_vals + sample(resid_vals)
##   }
## <environment: 0x00000079b4df7548>
```

```r
head(rlang::fn_env(boot_mtcars1)$fitted_vals)
```

```
## [1] 23.28261 21.91977 24.88595 20.10265 18.90014 18.79325
```

```r
head(rlang::fn_env(boot_mtcars1)$resid_vals)
```

```
## [1] -2.2826106 -0.9197704 -2.0859521  1.2973499 -0.2001440
## [6] -0.6932545
```

Questions?