Chapter 13: S3

Ryan Heslin

May 30, 2022

Basics

1.

`t.test` is the generic for the `t.test` function, which dispatches to the appropriate method. `t.data.frame` is the method of `t` for the `data.frame` class (it just coerces to matrix and invokes `NextMethod`).

2.

`data.frame` and many of the `as.*` and `is.*` family are major offenders.

3.

It coerces a data frame to a data frame, which means stripping classes inherited before `data.frame` and optionally adding row names. The overuse of dot separators makes it hard to understand that this is an S3 method, a problem that could have been solved by using snake case instead.

4.

The first dispatches to `mean.Date`, which coerces a date to integer, calls `mean` on it, and coerces back to `Date`. The second coerces to integer beforehand without doing this.

```r
set.seed(1014)
some_days <- as.Date("2017-01-31") + sample(10, 5)
```

```
mean(some_days)
```

```
[1] "2017-02-06"
```

```
#> [1] "2017-02-06"
mean(unclass(some_days))
```

```
[1] 17203.4
```

```
#> [1] 17203
```

5.

It generates the ECDF function for a given vector. The object contains the function and preserves the call used to create it.

```
x <- ecdf(rpois(100, 10))
x
```

```
Empirical CDF 
Call: ecdf(rpois(100, 10))
 x[1:18] =      2,      3,      4,  ...,     18,     19
```

```
#> Empirical CDF
#> Call: ecdf(rpois(100, 10))
#>  x[1:18] =  2,  3,  4,  ..., 2e+01, 2e+01
```

6.

A `table` object is an atomic vector array. Like arrays in general, it has dimensions and a `dimnames` attribute. The class is mostly used for its print method.

```
x <- table(rpois(100, 5))
x
```

```

 1  2  3  4  5  6  7  8  9 10 
 7  5 18 14 15 15 14  4  5  3 
```

Classes

1.

```r
data.frame2 <- function(..., row_names = NULL) {
  dots <- list(...)
  l_dots <- length(dots)
  if (l_dots == 0) {
    return(structure(list(), class = "data.frame"))
  }
  dots_names <- names(dots)
  has_names <- is.null(dots_names)
  col_names <- vector("character", l_dots)
  col_data <- vector("list", l_dots)
  unnamed <- rep(FALSE, l_dots)
  for (i in seq_along(dots)) {
    el <- dots[[i]]
    if ("dim" %in% attributes(el) && dim(el) > 1) {
      if (is.array(el)) el <- as.data.frame(el)
      el_rows <- nrow(el)
      if (has_names) {
        dots_names[[i]] <- el_name
        if (el_name == "") {
          unnamed[[i]] <- TRUE
        } else {
          colnames(el) <- paste(el_name, colnames(el), sep = "_")
        }
      }
    } else {
      el_rows <- length(el)
      if (has_names && (el_name <- dots_names[[i]]) == "") {
        unnamed[[i]] <- TRUE
      } else {
        col_names[[i]] <- el_name
      }
    }
    if (i == 1) {
      n_rows <- el_rows
    } else if (n_rows != el_rows) {
      stop("Number of rows mismatch")
    }
    col_data[[i]] <- el
  }
  # Validate row names (same length as data, unique)
  if (!is.null(row_names)) {
    if (!is.character(row_names)) row_names <- as.character(row_names)
    if (length(row_names) != n_rows) stop("Length of row names does not match data length")
    if (anyDuplicated(row_names)) stop("Duplicate row names")
  }
  # Supply default column names for unnamed arguments
  col_names[unnamed] <- paste0("V", seq_along(unnamed))
  names(col_data) <- el_names
  full_data <- do.call(c, col_data)

  structure(full_data, class = c("list", "data.frame"), row.names = row_names)
}
```

2.

I replicate the base behavior of replacing values absent from the levels with `NA` and excluding `NA` from the levels.

```
new_factor <- function(x = integer(), levels = character()) {
  stopifnot(is.integer(x))
  stopifnot(is.character(levels))
  levels <- levels[!is.na(levels)]

  structure(
    x,
    levels = levels,
    class = "factor"
  )
}

validate_factor <- function(x) {
  values <- unclass(x)
  levels <- attr(x, "levels")

  if (!all(values[!is.na(values)] > 0)) {
    stop(
      "All `x` values must be greater than zero",
      call. = FALSE
    )
  }


  if (length(levels) < max(values, na.rm = TRUE)) {
    stop(
      "There must be at least as many `levels` as possible values in `x`",
      call. = FALSE
    )
  }

  x
}


factor <- function(x = character(), levels = unique(x)) {
  levels <- as.character(levels)
  ind <- match(x, levels)
  validate_factor(new_factor(ind, levels))
}

factor(c("a", "a", "b"))
```

```
[1] a a b
Levels: a b
```

```
factor(1:3)
```

```
[1] 1 2 3
Levels: 1 2 3
```

```
factor(1:3, levels = c(1, 3))
```

```
[1] 1    <NA> 3
Levels: 1 3
```

```
factor(1:3, levels = "a")
```

```
[1] <NA> <NA> <NA>
Levels: a
```

```
#> [1] a a b
#> Levels: a b
```

3.

The base `factor` has the additional features of mapping different labels to the same levels and ordering the factors. More saliently, it assigns values that do not appear in the levels `NA` instead of throwing an error.

4.

5.

6.

The validator should confirm that inputs are integer vectors whose elements are all in [1, 3899], the range of valid Roman numerals, or character vectors of such valid Roman numerals. A constructor would then convert the input to integer, if necessary, then just set its class to `roman`, enabling the specialized methods to do their work.

`UseMethod` constructs a call by matching arguments in the generic's execution environment *as they came in*, forwarding them, then matching arguments defined in the execution environment. These are then forwarded to the method that is matched. So the redefinition of `x` is ignored because only the value passed is read.

Generics and Methods

#TODO

Object Styles

#TODO

Inheritance

```
g <- function(x) {
  x <- 10
  y <- 10
  UseMethod("g")
}
g.default <- function(x) c(x = x, y = y)

x <- 1
y <- 1
g(x)
```

```
 x  y
 1 10
```

```
#>  x  y
#>  1 10
```

`NextMethod` skips the first method matched by `UseMethod` and continues searching, potentially to internal generics. Arguments are passed as promises to evaluate in the caller environment of NextMethod.

This makes it possible to force use of a default or internal method by placing a call to `NextMethod` in a class-specific method.

```
new_secret <- function(x, ..., class = character()) {
  stopifnot(is.double(x))

  structure(
    x,
    ...,
    class = c(class, "secret")
  )
}
new_supersecret <- function(x) {
  new_secret(x, class = "supersecret")
}

print.supersecret <- function(x, ...) {
  print(rep("xxxxx", length(x)))
  invisible(x)
}

x2 <- new_supersecret(c(15, 1, 456))
x2
```

```
[1]  15   1 456
attr(,"class")
[1] "supersecret" "secret"
```

```
x <- structure(1:10, class = "test")
t(x)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1    2    3    4    5    6    7    8    9
     [,10]
[1,]    10
attr(,"class")
[1] "test"
```

1.

`[.Date` delegates to `NextMethod`, passing the most specific class of the argument `x` as determined by `oldClass`. This means a `Date` subclass `foo` is dispatched to `[.foo`.

```
library(sloop)
s3_methods_generic("[")
```

generic	class	visible	source
[	acf	FALSE	registered S3method
[	arrow	FALSE	registered S3method
[	AsIs	TRUE	base
[	bench_bytes	FALSE	registered S3method
[	bench_expr	FALSE	registered S3method
[	bench_mark	FALSE	registered S3method
[	bench_time	FALSE	registered S3method
[	bibentry	FALSE	registered S3method
[	cell_addr	FALSE	registered S3method
[	check_details_changes	FALSE	registered S3method
[	cli_doc	FALSE	registered S3method
[	data.frame	TRUE	base
[	Date	TRUE	base
[	difftime	TRUE	base
[	Dlist	TRUE	base
[	DLLInfoList	TRUE	base
[	factor	TRUE	base
[	formula	FALSE	registered S3method
[	fs_bytes	FALSE	registered S3method
[	fs_path	FALSE	registered S3method
[	fs_perms	FALSE	registered S3method
[	fseq	FALSE	registered S3method
[	fun_list	FALSE	registered S3method
[	GenericSummary	FALSE	registered S3method
[	getAnywhere	FALSE	registered S3method
[	gList	FALSE	registered S3method
[	glue	FALSE	registered S3method
[	gpar	FALSE	registered S3method
[	grouped_df	FALSE	registered S3method
[	gtable	FALSE	registered S3method
[	hexmode	TRUE	base
[	insensitive	FALSE	registered S3method
[	listof	TRUE	base
[	lobstr_bytes	FALSE	registered S3method
[	mapped_discrete	FALSE	registered S3method
[	news_db	FALSE	registered S3method
[	noquote	TRUE	base
[	numeric_version	TRUE	base
[	octmode	TRUE	base
[	person	FALSE	registered S3method
[	POSIXct	TRUE	base
[	POSIXlt	TRUE	base
[	quosure	FALSE	registered S3method
[	quosures	FALSE	registered S3method
[	raster	FALSE	registered S3method
[	rlang_ctxt_pronoun	FALSE	registered S3method
[	rlang_data_pronoun	FALSE	registered S3method
[	rlang_envs	FALSE	registered S3method
[	rlang:::list_of_conditions	FALSE	registered S3method
[	rlib_bytes	FALSE	registered S3method
[	roman	FALSE	registered S3method
[	rowwise_df	FALSE	registered S3method

generic	class	visible	source
[	shingle	FALSE	registered S3method
[	simple.list	TRUE	base
[	spec_tbl_df	FALSE	registered S3method
[	SQL	FALSE	registered S3method
[	table	TRUE	base
[	tbl_df	FALSE	registered S3method
[	terms	FALSE	registered S3method
[	trellis	FALSE	registered S3method
[	ts	FALSE	registered S3method
[	tskernel	FALSE	registered S3method
[	uneval	FALSE	registered S3method
[	unit	FALSE	registered S3method
[	vctrs_rcrd	FALSE	registered S3method
[	vctrs_sclr	FALSE	registered S3method
[	vctrs_unspecified	FALSE	registered S3method
[	vctrs_vctr	FALSE	registered S3method
[	vpPath	FALSE	registered S3method
[	warnings	TRUE	base
[	xml_missing	FALSE	registered S3method
[	xml_nodeset	FALSE	registered S3method

2.

It looks like `POSIXct` methods are more verbose and do more elaborate checking involving timezones. `print` is naturally the same for both.

3.

`generic2` dispatches on the class of `x`. `generic2.b` is called first, the class reassignment is ignored, then `NextMethod` dispatches to `generic.a2`.

```r
generic2 <- function(x) UseMethod("generic2")
generic2.a1 <- function(x) "a1"
generic2.a2 <- function(x) "a2"
generic2.b <- function(x) {
  class(x) <- "a1"
  NextMethod()
}

generic2(structure(list(), class = c("b", "a2")))
```

```
[1] "a2"
```

Dispatch Details

1.

Internal methods dispatch only on implicit class (what `1:5` has), not explicit class set by `class`.

2.

`Math.data.frame` checks types before using `lapply` to compute the operation. `Math.difftime` records units before forwarding to `NextMethod`. The `factor` and `PosixLT` methods warn the user that calling them is nonsensical.

```
sloop::s3_methods_generic("Math")
```

generic	class	visible	source
Math	data.frame	TRUE	base
Math	Date	TRUE	base
Math	difftime	TRUE	base
Math	factor	TRUE	base
Math	POSIXt	TRUE	base
Math	quosure	FALSE	registered S3method
Math	vctrs_sclr	FALSE	registered S3method
Math	vctrs_vctr	FALSE	registered S3method

3.

It tracks units and throws an error for unsupported operations.