Debugging in R

01-23-2020

Overall approach

- Google first
- Make it repeatable
    - set random seed using `set.seed()`
- Figure out where it is
- Fix it and test it

Different types of erros

- syntax / parsing error
- runtime error

Locating errors

```
f <- function(a) g(a)
g <- function(b) h(b)
h <- function(c) i(c)
i <- function(d) {
  if (!is.numeric(d)) {
    stop("`d` must be numeric", call. = FALSE)
  }
  d + 10
}
f("a")
```

See also `debug.R` demonstration.

Lazy evaluation

```
j <- function() k()
k <- function() stop("Oops!", call. = FALSE)
f(j())
```

```
rlang::with_abort(f(j()))
rlang::last_trace()
```

Using `recover()`

By setting ,

```r
options(error = recover)
```

a interactive prompt will be displayed that you get an error.

Interactive debugger in RStudio

When we encounter an error, we could hit `Rerun with Debug` in RStudio to start interactive debugging.

```r
f("a")
```

Sometimes, we know something is definitely wrong but the code runs fine.

See `debug2.R` for demonstration.

Interactive debugger elsewhere

`browser()`

When `browser()` is run, a interactive prompt will be shown.

```r
g <- function(b) {
  browser()
  h(b)
}
g(10)
```

`browser()` is a regular function call which means that you can run it conditionally

```r
g <- function(b) {
  if (b < 0) {
    browser()
  }
  h(b)
}
g(10)
```

See `debug2.R` for demonstration.

`debug()` and `debugonce()`

`debug` takes a single argument, the name of a function. When you pass the name of a function to debug, that function is flagged for debugging.

See `debug3.R` for demonstration.

See `debug4.R` for demonstration.

Debug R batch scripts

See `debug5.R` for demonstration.

```r
# In a later interactive session ----
load("last.dump.rda")
debugger()
```

Reference

Advanced R https://adv-r.hadley.nz/debugging.html