R language definition

Shaofeng Duan

Excerpt from [R langugage Definition](https://cran.r-project.org/doc/manuals/r-release/R-lang.html#Basic-types)

## Objects

R does not provide **direct access to computer's memory** but rather provides a number of specializxed data structures we will refer to as objects.

The return value of typeof function:

|  |  |
| --- | --- |
| Value | Meaning |
| "NULL" | NULL |
| "symbol" | a variable name |
| "pairlist" | a pairlist object(mainly internal) |
| "closure" | a function |
| "environment" | an environment |
| "promise" | an object used to implement lazy evaluation |
| "language" | an R language construct |
| "special" | an internal function that does not evaluate its arguments |
| "builtin" | an internal function that evaluates its arguments |
| "expression" | an expression object |
| ... | general values are not presented... |

R objects are often coerced to different types during computations.

### Basic types

R has **six** basic vector types: logcial, integer, real, complex, string, raw.

#### Language objects

There are three types of object that constitute the R language: *calls, expressions, names*.

##### Function objects

Three components of function: a formal arguments list, a body and an environment, which can be manipulated by three constructs: *formals, body, environment*. The function can be converted to list with as.list.

#### Dot-dot-dot

The '...' is stored as a type of pairlist, which can be captured as a list,

args <- list(...)  
for(a in args)  
{  
   
}

### Attributes

All objects except for NULL have one or more attributes attached to them.

### Special compound objects

* factor
* data frame

## Evaluation of expressions

When a user types a command at the prompt (or when an expression is read from a file) the first thing that happens to it is that the command is transformed by the parser into an internal representation. The evaluator executes parsed R expressions and returns the value of the expression. All expressions have a value. **This is the core of language.**

### Simple evaluation

#### Constants

typeof(1)

## [1] "double"

typeof(1L)

## [1] "integer"

#### Function calls

A special function calls is that the function appears on the left hand side, such as:

class(x) <- "foo"

Which acturally calls the function class<-, and the value of argument is modified and the modified version is then stored back into the original variable.

#### Operators

x+y is equivalent with `+`(x, y)

## Functiosn

### Evaluation

#### Argument matching

This is done by a three-pass process: 1. Exact matching on tags. 1. Partial matching on tags. 1. Positional matching.

#### Argument evaluation

## For the importance of following sentences, I translated them into chinese.

对函数的参数求值需要注意的最重要的一点是对默认参数与提供参数的求值方式是不同的。直接提供的参数是在调用环境中进行求值，而默认参数是在函数的执行环境中进行的。 R中函数参数传递的方式为值传递。

test\_func1 <- function(x, y = x + 1)  
{  
 y  
 x <- x + 1  
 return(y)  
}  
  
test\_func2 <- function(x, y = x + 1)  
{  
 x <- x + 1  
 return(y)  
}

## [1] "evaluated before x value changed, the y is 3"

## [1] "evaluated after x value changed, the y is 4"

## Object-oriented programming

## UseMethod

UseMethod can only be called from the body of a function. It is an interesting process for dispatching the method, It firstly to find the accuracy method called the name consisting of the name of the generic function and the class of the first argument, if not exist, it keeps looking for the method called the combined name with generic function and the second class of the arguments. This procedure continues until calling the method called <name of generic function>.default. After finding the right method, it execute that method in the evaluation environment of the UseMethod.

## Computing on the language

### Direct manipulation of language objects

There are three kinds of language objects that are available for modification, *calls, expressions, functions*. quote is used to get a call object.

e1 <- quote(2 + 2)  
e2 <- quote(plot(x, y))  
e1

## 2 + 2

e2

## plot(x, y)

e2[[1]] <- as.name("+")  
e2

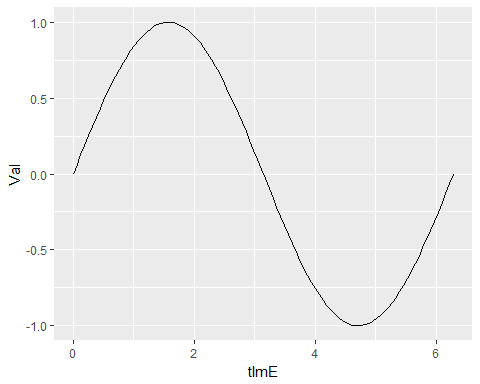
## x + y

The arguments are not evaluated, the result is simply the parsed argument.

### Substitutions

A simple example:

library(ggplot2)  
  
plot\_time\_series <- function(x, y, df)  
{  
 aa <- match.call()  
 p <- eval(substitute(ggplot(data = df, aes(x = ix,   
 y = iy)), list(ix = aa[[2]],   
 iy = aa[[3]])))  
}  
  
x <- seq(0, 2\*pi, length.out = 100)  
y <- sin(x)  
  
df <- data.frame(tImE = x, Val = y)  
  
plot\_time\_series(tImE, Val, df) + geom\_line()



### Evaluation of expression objects

A funny example:

eval(substitute(mode(x), list(x = quote(2 + 2))))

## [1] "numeric"

eval(substitute(mode(x), list(x = expression(2 + 2))))

## [1] "expression"

The difference dues to the reason that call objects are indistinguishable from subexpressions in a parse tree.

### Manipulation of functions calls

sys.call function just returns the real calling situation, while match.call returns the actural bounding situation. Watching the following example:

f <- function(x, y, ...) sys.call()  
f(y = 1, 2, z = 3, 4)

## f(y = 1, 2, z = 3, 4)

f1 <- function(x, y, ...) match.call()  
f1(y = 1, 2, z = 3, 4)

## f1(x = 2, y = 1, z = 3, 4)

## Exception handling