Package: quotedargs (via r-universe)

September 7, 2024

Type Package

Title A Way of Writing Functions that Quote their Arguments

Version 0.1.3

Author Radford Neal

Maintainer Radford Neal <radfordneal@gmail.com>

Description A facility for writing functions that quote their
arguments, may sometimes evaluate them in the environment where
they were quoted, and may pass them as quoted to other
functions.

License GPL-2 | GPL-3

LazyData TRUE

NeedsCompilation yes

Repository https://radfordneal.r-universe.dev

RemoteUrl https://github.com/radfordneal/quotedargs

RemoteRef HEAD

RemoteSha f9a10b96ce147785fec1e6074f86660332787019

Contents

---

quotedargs-package   *Facility for using quoted arguments*

Description

This package assists with writing functions that automatically quote their arguments, but that may also wish to evalute them, in their original environment. These quoted arguments can be passed to other functions that quote their arguments, with proper passing of their quoting environment. It is also possible to set up a variable so that it looks just like a quoted argument.

In the simplest applications of this package, one can think of calling `quoted_arg(x)` (see below) as altering the default meaning of a reference to an argument x from the value of the actual argument, with the expression passed for x accessible via `substitute(x)`, so that instead the default is the expression passed for x, with the value accessible via `quoted_eval(x)`, as described below.

However, in more complex applications, the facilities provided by this package are more than just a convenient change of defaults, as they allow functions that quote arguments to be combined in ways that would otherwise be difficult.

Usage

```
quoted_arg (...)
quoted_eval (arg)
quoted_environment (arg)
notquoted (x)

quoted_assign (name, value, eval.env, assign.env = parent.frame())
```

Arguments

`...`	names (unquoted) of function arguments that should be quoted
`arg`	the name of a quoted function argument (unquoted)
`x`	any expression.
`name`	the name (as a character string or symbol) of a variable to assign to
`value`	a value to assign to the variable `name`
`eval.env`	the environment in which `value` may be evaluated; may be missing, with default as described below
`assign.env`	the environment in which to assign to `name`

Details

The `quoted_arg` function should be called at the start of a function that uses quoted arguments, with arguments that are the (unquoted) names of the arguments that should be quoted. After the call of `quoted_arg`, simple references to these arguments will give the expressions passed as arguments, rather than the values of these expressions. Currently, `...`, `..1`, `..2`, etc. are not allowed as arguments of `quoted_arg`.

The caller of a function can disable any quoting with `quoted_arg` by passing `notquoted(x)` instead of x, in which case x will be evaluated when `quoted_arg` is called, and references to x will deliver this value, not the expression.

To obtain the value of a quoted argument, `quoted_eval` can be used. The evaluation will be done in the environment of the quoted expression. If `quoted_eval` is called more than once for the same

argument, the argument will be evaluated that many times (possibly with different results). If the actual argument used `notquoted`, `quoted_eval` will simply return the already-evaluated argument.

The environment used by `quoted_eval` can be obtained with `quoted_environment`, which will be `NULL` if the actual argument used `notquoted`, and may be `emptyenv()` if the expression is self-evaluating, and hence its evaluation would not reference an environment.

When a quoted argument is passed as an argument to another function that quotes that argument, the quoted argument received will be the argument originally passed, not a quoting of the name of the quoted argument.

A variable can be set up so that it looks like a quoted argument using `quoted_assign`.

The name of the variable to set is specified by the `name` argument of `quoted_assign`, which must evaluate to a single character string or a symbol. The environment in which this variable is assigned is specified by the `assign.env` argument, which defaults to the current environment (the parent frame of `quoted_assign`).

The `value` argument to `quoted_assign` is evaluated to obtain an expression analogous to an actual argument, which is stored in the variable specified by `name`. The environment `eval.env` is stored with the assigned expression (in a "promise"), and will be used when evaluating this expression if `quoted_eval` is called for the assigned variable. If `eval.env` is missing, it defaults to the current environment, unless `value` is itself a quoted argument, in which case the default is `quoted_environment(value)`. If the `eval.env` argument of `quoted_assign` is `NULL`, what is stored in `name` will look like a quoted argument in which the actual argument used `notquoted`, and evaluated to `value`, with the expression stored in the promise being the unevaluated form of `value`.

Value

`quoted_eval` and `quoted_environment` return values as described above.

`notquoted` returns its argument.

`quoted_arg` and `quoted_assign` always return NULL.

See Also

[substitute](#), for how to get at the expression passed when an argument is not quoted.

[delayedAssign](#), for another function that is somewhat analogous to `quoted_assign`.

Examples

```
# A simple example in which both the expression passed and its value
# are used.

showmean <- function (v) {
    quoted_arg(v)
    cat ("Mean of", deparse(v), "is", mean(quoted_eval(v)), "\n")
}

showmean(100+(1:3))                 # Will print 100 + (1:3)
showmean(notquoted(100+(1:3)))      # Will print c(101, 102, 103)

# A function that uses the function above, passing along its quoted
# argument.
```

```
showmeansummary <- function (u) {
    quoted_arg(u)
    cat("Summary: ")
    showmean(u)
}

showmeansummary(100+(1:3))              # Will print 100 + (1:3), not u!
showmeansummary(notquoted(100+(1:3))) # Will print c(101, 102, 103)

u <- v <- 100+(1:3)    # Evaluation of showmeansummary's argument
showmeansummary(u)     #   is done in the environment of the caller,
showmeansummary(v)     #   not that of showmeansummary or showmean


# An illustration of quoted arguments being evaluated many times.

prsim <- function (a,b,n) {
    quoted_arg(a,b)
    cat ("Running simulation to find probability that all\n")
    cat (deparse(a), "are greater than all", deparse(b), "\n")
    count <- 0
    for (i in 1:n) {
        if (min(quoted_eval(a)) > max(quoted_eval(b)))
            count <- count + 1
    }
    count / n
}

set.seed(1)
prsim (rexp(10,0.1), rnorm(10,1), 1000)


# Creating a variable that behaves like a quoted argument.

quoted_assign("x",quote(runif(1)))

set.seed(1)
cat (paste0("Two evaluations of ",deparse(x),": "),
     quoted_eval(x), quoted_eval(x),
     "\n")


# Examples of when quotation of an argument is passed on.

qfun1 <- function (x) { quoted_arg(x); list(x,quoted_eval(x)) }

qfun2 <- function (y) {
    quoted_arg(y)
    a <- y
    quoted_assign ("b", y)
    list(qfun1(y),qfun1((y)),qfun1(a),qfun1(b))
}
```

```
qfun2(1+2)


# Example of how quoted_arg and quoted_eval can be used to avoid
# copying of a large object.

sum_first_last1 <- function (v) {
    v[1] + v[length(v)]
}

sum_first_last2 <- function (v) {
    quoted_arg(v)
    quoted_eval(v)[1] + quoted_eval(v)[length(quoted_eval(v))]
}

f <- function (sumfl) {
    x <- 1:100000
    r <- sumfl(x)
    x[2] <- 0L
    r
}

f(sum_first_last1)  # x[2] <- 0L first copies x (in current R implementations)
f(sum_first_last2)  # x[2] <- 0L does not result in x being copied


# Example of using quotedargs to build functions that take as
# arguments expressions that may reference columns of a data
# frame and variables accessible in the caller's environment.
# The data frame columns take precedence, except that the data
# frame is skipped for expressions enclosed in O(...).

dfeval <- function (df, expr) {     # Find value of expression
    quoted_arg(expr)
    env <- new.env (parent = quoted_environment(expr), hash=FALSE)
    env$O <- function (z) { quoted_arg(z); eval(z,parent.env(environment())) }
    environment(env$O) <- env
    eval (expr, df, env)
}

dfchange <- function (df, expr) {  # Return data frame changed by assignments
    quoted_arg(expr)
    env <- new.env (parent = quoted_environment(expr), hash=FALSE)
    env$O <- function (z) { quoted_arg(z); eval(z,parent.env(environment())) }
    environment(env$O) <- env
    dfenv <- as.environment(df)
    parent.env(dfenv) <- env
    eval (expr, dfenv)
    as.data.frame (as.list (dfenv))
}

dfchange_var <- function (df, expr) {  # Actually change df variable passed
```

```
    quoted_arg(df,expr)
    newdf <- dfchange (quoted_eval(df), expr)
    assign (as.character(df), newdf, quoted_environment(df))
}

tstdf <- as.data.frame (list (x = 1:4, y = c("a","b","c","d")))

y <- 100
dfeval (tstdf, paste0(y,x))
dfeval (tstdf, x * O(y))

dfchange (tstdf, { z <- 10*x; x <- x + O(y) })

dfchange_var (tstdf, x <- 1000+x)
tstdf
```

Index