

Package ‘lazyeval’

January 27, 2015

Version 0.1.10

Title Lazy (Non-Standard) Evaluation

Description A disciplined approach to non-standard evaluation.

License GPL-3

LazyData true

Depends R (>= 3.1.0)

Suggests knitr, rmarkdown (>= 0.2.65), microbenchmark, testthat

VignetteBuilder knitr

Author Hadley Wickham [aut, cre],
RStudio [cph]

Maintainer Hadley Wickham <hadley@rstudio.com>

NeedsCompilation yes

Repository CRAN

Date/Publication 2015-01-02 18:01:12

R topics documented:

as.lazy	2
interp	2
lazy_	3
lazy_dots	4
lazy_eval	5
make_call	6
missing_arg	7
Index	8

<code>as.lazy</code>	<i>Convert an object to a lazy expression or lazy dots.</i>
----------------------	---

Description

Convert an object to a lazy expression or lazy dots.

Usage

```
as.lazy(x, env = baseenv())
```

```
as.lazy_dots(x, env)
```

Arguments

<code>x</code>	An R object. Current methods for <code>as.lazy()</code> convert formulas, character vectors, calls and names. Methods for <code>as.lazy_dots()</code> convert lists and character vectors (by calling <code>lapply()</code> with <code>as.lazy()</code> .)
<code>env</code>	Environment to use for objects that don't already have associated environment.

Examples

```
as.lazy(~ x + 1)
as.lazy(quote(x + 1), globalenv())
as.lazy("x + 1", globalenv())

as.lazy_dots(list(~x, y = ~z + 1))
as.lazy_dots(c("a", "b", "c"), globalenv())
as.lazy_dots(~x)
as.lazy_dots(quote(x), globalenv())
as.lazy_dots(quote(f()), globalenv())
as.lazy_dots(lazy(x))
```

<code>interp</code>	<i>Interpolate values into an expression.</i>
---------------------	---

Description

This is useful if you want to build an expression up from a mixture of constants and variables.

Usage

```
interp(`_obj`, ..., .values)
```

Arguments

`_obj` An object to modify: can be a call, name, formula, [lazy](#), or a string.
`..., .values` Either individual name-value pairs, or a list (or environment) of values.

Examples

```
# Interp works with formulas, lazy objects, quoted calls and strings
interp(~ x + y, x = 10)
interp(lazy(x + y), x = 10)
interp(quote(x + y), x = 10)
interp("x + y", x = 10)

# Use as.name if you have a character string that gives a
# variable name
interp(~ mean(var), var = as.name("mpg"))
# or supply the quoted name directly
interp(~ mean(var), var = quote(mpg))

# Or a function!
interp(~ f(a, b), f = as.name("+"))
# Remember every action in R is a function call:
# http://adv-r.had.co.nz/Functions.html#all-calls

# If you've built up a list of values through some other
# mechanism, use .values
interp(~ x + y, .values = list(x = 10))

# You can also interpolate variables defined in the current
# environment, but this is a little risky.
y <- 10
interp(~ x + y, .values = environment())
```

lazy_

Capture expression for later lazy evaluation.

Description

`lazy()` uses non-standard evaluation to turn promises into lazy objects; `lazy_()` does standard evaluation and is suitable for programming.

Usage

```
lazy_(expr, env)
```

```
lazy(expr, env = parent.frame(), .follow_symbols = TRUE)
```

Arguments

<code>expr</code>	Expression to capture. For <code>lazy_</code> must be a name or a call.
<code>env</code>	Environment in which to evaluate <code>expr</code> .
<code>.follow_symbols</code>	If TRUE, the default, follows promises across function calls. See <code>vignette("chained-promises")</code> for details.

Details

Use `lazy()` like you'd use `substitute()` to capture an unevaluated promise. Compared to `substitute()` it also captures the environment associated with the promise, so that you can correctly replay it in the future.

Examples

```

lazy_(quote(a + x), globalenv())

# Lazy is designed to be used inside a function - you should
# give it the name of a function argument (a promise)
f <- function(x = b - a) {
  lazy(x)
}
f()
f(a + b / c)

# Lazy also works when called from the global environment. This makes
# easy to play with interactively.
lazy(a + b / c)

# By default, lazy will climb all the way back to the initial promise
# This is handy if you have if you have nested functions:
g <- function(y) f(y)
h <- function(z) g(z)
f(a + b)
g(a + b)
h(a + b)

# To avoid this behaviour, set .follow_symbols = FALSE
# See vignette("chained-promises") for details

```

lazy_dots

Capture ... (dots) for later lazy evaluation.

Description

Capture ... (dots) for later lazy evaluation.

Usage

```
lazy_dots(..., .follow_symbols = FALSE)
```

Arguments

```
...           Dots from another function
.follow_symbols
              If TRUE, the default, follows promises across function calls. See vignette("chained-promises")
              for details.
```

Value

A named list of [lazy](#) expressions.

Examples

```
lazy_dots(x = 1)
lazy_dots(a, b, c * 4)

f <- function(x = a + b, ...) {
  lazy_dots(x = x, y = a + b, ...)
}
f(z = a + b)
f(z = a + b, .follow_symbols = TRUE)

# .follow_symbols is off by default because it causes problems
# with lazy loaded objects
lazy_dots(letters)
lazy_dots(letters, .follow_symbols = TRUE)

# You can also modify a dots like a list. Anything on the RHS will
# be coerced to a lazy.
l <- lazy_dots(x = 1)
l$y <- quote(f)
l[c("y", "x")]
l["z"] <- list(~g)

c(lazy_dots(x = 1), lazy_dots(f))
```

lazy_eval

Evaluate a lazy expression.

Description

Evaluate a lazy expression.

Usage

```
lazy_eval(x, data = NULL)
```

Arguments

x	A lazy object or a formula.
data	Option, a data frame or list in which to preferentially look for variables before using the environment associated with the lazy object.

Examples

```
f <- function(x) {
  z <- 100
  ~ x + z
}
z <- 10
lazy_eval(f(10))
lazy_eval(f(10), list(x = 100))
lazy_eval(f(10), list(x = 1, z = 1))

lazy_eval(lazy_dots(a = x, b = z), list(x = 10))
```

make_call

Make a call with lazy_dots as arguments.

Description

In order to exactly replay the original call, the environment must be the same for all of the dots. This function circumvents that a little, falling back to the `baseenv()` if all environments aren't the same.

Usage

```
make_call(fun, args)
```

Arguments

fun	Function as symbol or quoted call.
args	Arguments to function; must be a lazy_dots object, or something <code>as.lazy_dots()</code> can coerce..

Value

A list:

env	The common environment for all elements
expr	The expression

Examples

```
make_call(quote(f), lazy_dots(x = 1, 2))
make_call(quote(f), list(x = 1, y = ~x))
make_call(quote(f), ~x)

# If no known or no common environment, fails back to baseenv()
make_call(quote(f), quote(x))
```

missing_arg	<i>Generate a missing argument.</i>
-------------	-------------------------------------

Description

Generate a missing argument.

Usage

```
missing_arg()
```

Examples

```
interp(~f(x), x = missing_arg())
```

Index

`as.lazy`, 2
`as.lazy_dots`, 6
`as.lazy_dots (as.lazy)`, 2

`baseenv`, 6

`interp`, 2

`lapply`, 2
`lazy`, 3, 5
`lazy (lazy_)`, 3
`lazy_`, 3
`lazy_dots`, 4
`lazy_eval`, 5

`make_call`, 6
`missing_arg`, 7

`substitute`, 4