Package 'memoise'

August 29, 2016

Encoding UTF-8

Title Memoisation of Functions

Version 1.0.0

Description Cache the results of a function so that when you call it again with the same arguments it returns the pre-computed value.

URL https://github.com/hadley/memoise

BugReports https://github.com/hadley/memoise/issues

Imports digest (>= 0.6.3)

Suggests testthat

License MIT + file LICENSE

RoxygenNote 5.0.1

NeedsCompilation no

Author Hadley Wickham [aut],
Jim Hester [aut, cre],
Kirill Müller [aut]

Maintainer Jim Hester <jim.hester@rstudio.com>

Repository CRAN

Date/Publication 2016-01-29 05:58:01

R topics documented:

forget 2
has_cache 2
is.memoised 3
memoise 4
timeout 6

Index **7**

forget *Forget past results. Resets the cache of a memoised function.*

Description

Forget past results. Resets the cache of a memoised function.

Usage

```
forget(f)
```

Arguments

f	memoised function

See Also

memoise, is.memoised

Examples

```
memX <- memoise(function() { Sys.sleep(1); runif(1) })
# The forget() function
system.time(print(memX()))
system.time(print(memX()))
forget(memX)
system.time(print(memX()))
```

has_cache *Test whether a memoised function has been cached for particular arguments.*

Description

Test whether a memoised function has been cached for particular arguments.

Usage

```
has_cache(f, ...)
```

Arguments

f	Function to test.
...	arguments to function.

See Also

is.memoised, memoise

Examples

```
mem_sum <- memoise(sum)
has_cache(mem_sum)(1, 2, 3) # FALSE
mem_sum(1, 2, 3)
has_cache(mem_sum)(1, 2, 3) # TRUE
```

`is.memoised` *Test whether a function is a memoised copy. Memoised copies of functions carry an attribute* `memoised = TRUE`*, which is.memoised() tests for.*

Description

Test whether a function is a memoised copy. Memoised copies of functions carry an attribute `memoised = TRUE`, which is.memoised() tests for.

Usage

```
is.memoised(f)
```

Arguments

`f`	Function to test.

See Also

memoise, forget

Examples

```
mem_lm <- memoise(lm)
is.memoised(lm) # FALSE
is.memoised(mem_lm) # TRUE
```

`memoise` *Memoise a function.*

Description

`mf <- memoise(f)` creates `mf`, a memoised copy of `f`. A memoised copy is basically a lazier version of the same function: it saves the answers of new invocations, and re-uses the answers of old ones. Under the right circumstances, this can provide a very nice speedup indeed.

Usage

```
memoise(f, ..., envir = environment(f))
```

Arguments

`f`	Function of which to create a memoised copy.
`...`	optional variables specified as formulas with no RHS to use as additional restrictions on caching. See Examples for usage.
`envir`	Environment of the returned function.

Details

There are two main ways to use the `memoise` function. Say that you wish to memoise `glm`, which is in the `stats` package; then you could use
`mem_glm <- memoise(glm)`, or you could use
`glm <- memoise(stats::glm)`.
The first form has the advantage that you still have easy access to both the memoised and the original function. The latter is especially useful to bring the benefits of memoisation to an existing block of R code.

Two example situations where `memoise` could be of use:

- You're evaluating a function repeatedly over the rows (or larger chunks) of a dataset, and expect to regularly get the same input.
- You're debugging or developing something, which involves a lot of re-running the code. If there are a few expensive calls in there, memoising them can make life a lot more pleasant. If the code is in a script file that you're `source()`ing, take care that you don't just put
  `glm <- memoise(stats::glm)`
  at the top of your file: that would reinitialise the memoised function every time the file was sourced. Wrap it in
  `if (!is.memoised(glm))` , or do the memoisation call once at the R prompt, or put it somewhere else where it won't get repeated.

See Also

`forget`, `is.memoised`, `timeout`, `http://en.wikipedia.org/wiki/Memoization`

Examples

```
# a() is evaluated anew each time. memA() is only re-evaluated
# when you call it with a new set of parameters.
a <- function(n) { runif(n) }
memA <- memoise(a)
replicate(5, a(2))
replicate(5, memA(2))

# Caching is done based on parameters' value, so same-name-but-
# changed-value correctly produces two different outcomes...
N <- 4; memA(N)
N <- 5; memA(N)
# ... and same-value-but-different-name correctly produces
#     the same cached outcome.
N <- 4; memA(N)
N2 <- 4; memA(N2)

# memoise() knows about default parameters.
b <- function(n, dummy="a") { runif(n) }
memB <- memoise(b)
memB(2)
memB(2, dummy="a")
# This works, because the interface of the memoised function is the same as
# that of the original function.
formals(b)
formals(memB)
# However, it doesn't know about parameter relevance.
# Different call means different cacheing, no matter
# that the outcome is the same.
memB(2, dummy="b")

# You can create multiple memoisations of the same function,
# and they'll be independent.
memA(2)
memA2 <- memoise(a)
memA(2)  # Still the same outcome
memA2(2) # Different cache, different outcome

# Don't do the same memoisation assignment twice: a brand-new
# memoised function also means a brand-new cache, and *that*
# you could as easily and more legibly achieve using forget().
# (If you're not sure whether you already memoised something,
#  use is.memoised() to check.)
memA(2)
memA <- memoise(a)
memA(2)
# Making a memoized automatically time out after 10 seconds.
memA3 <- memoise(a, ~{current <- as.numeric(Sys.time()); (current - current %% 10) %/% 10 })
memA3(2)

# The timeout function is any easy way to do the above.
memA4 <- memoise(a, ~timeout(10))
```

```
memA4(2)
```

`timeout` *Return a new number after a given number of seconds*

Description

This function will return a number corresponding to the system time and remain stable until a given number of seconds have elapsed, after which it will update to the current time. This makes it useful as a way to timeout and invalidate a memoised cache after a certain period of time.

Usage

```
timeout(seconds, current = as.numeric(Sys.time()))
```

Arguments

`seconds`	Number of seconds after which to timeout.
`current`	The current time as a numeric.

Value

A numeric that will remain constant until the seconds have elapsed.

See Also

`memoise`

Examples

```
a <- function(n) { runif(n) }
memA <- memoise(a, ~timeout(10))
memA(2)
```

Index

forget, 2, *3, 4*

has_cache, 2

is.memoised, *2, 3*, 3, *4*
is.memoized *(*is.memoised*)*, 3

memoise, *2, 3*, 4, *6*
memoize *(*memoise*)*, 4

timeout, *4*, 6