Package ‘memoise’

April 21, 2017

Encoding UTF-8
Title Memoisation of Functions
Version 1.1.0
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, aws.s3, httr, covr
Additional_repositories http://cloudyr.github.io/drat
License MIT + file LICENSE
RoxygenNote 6.0.1
NeedsCompilation no
Author Hadley Wickham [aut],
Jim Hester [aut, cre],
Kirill Müller [aut],
Daniel Cook [aut]
Maintainer Jim Hester <jim.hesterrstudio.com>
Repository CRAN
Date/Publication 2017-04-21 05:54:22 UTC

R topics documented:
cache_filesystem ........................................ 2
cache_memory ............................................ 2
cache_s3 .................................................. 3
forget ....................................................... 3
has_cache .................................................. 4
is.memoised .............................................. 5
memoise ................................................... 5
timeout .................................................... 7
cache_filesystem  

**Description**

Use a cache on the local filesystem that will persist between R sessions.

**Usage**

```r
cache_filesystem(path, algo = "xxhash64")
```

**Arguments**

- `path`: Directory in which to store cached items.
- `algo`: The hashing algorithm used for the cache, see `digest` for available algorithms.

**Examples**

```r
## Not run:
# Use with Dropbox

db <- cache_filesystem("~/Dropbox/.rcache")

mem_runif <- memoise(runif, cache = db)

# Use with Google Drive

gd <- cache_filesystem("~/Google Drive/.rcache")

mem_runif <- memoise(runif, cache = gd)

## End(Not run)
```

cache_memory

**Description**

A cache in memory, that lasts only in the current R session.

**Usage**

```r
cache_memory(algo = "sha512")
```
Arguments
algo The hashing algorithm used for the cache, see digest for available algorithms.

Amazon Web Services S3 Cache Amazon Web Services S3 backed cache, for remote caching.

Description
Amazon Web Services S3 Cache Amazon Web Services S3 backed cache, for remote caching.

Usage
cache_s3(cache_name, algo = "sha512")

Arguments
cache_name Bucket name for storing cache files.
algo The hashing algorithm used for the cache, see digest for available algorithms.

Examples

## Not run:
# Set AWS credentials.
Sys.setenv("AWS_ACCESS_KEY_ID" = "<access key>",
    "AWS_SECRET_ACCESS_KEY" = "<access secret>")

# Set up a unique bucket name.
s3 <- cache_s3("unique-bucket-name")
mem_runif <- memoise(runif, cache = s3)

## End(Not run)

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)
has_cache

Arguments

f memoised function

See Also

memoise, is.memoised

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

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

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()))
is.memoised

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), cache = cache_memory())

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.
cache
Cache 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

\[ \text{mem glm} \leftarrow \text{memoise} (\text{glm}), \text{or you could use} \]

\[ \text{glm} \leftarrow \text{memoise} (\text{stats::glm}). \]

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

\[ \text{glm} \leftarrow \text{memoise} (\text{stats::glm}) \]

at the top of your file: that would reinitialise the memoised function every time the file was sourced. Wrap it in

\[ \text{if} \left( \text{is.memoised} (\text{glm}) \right) \], or do the memoisation call once at the R prompt, or put it somewhere else where it won’t get repeated.

See Also

\[ \text{forget, is.memoised, timeout, http://en.wikipedia.org/wiki/Memoization} \]

Examples

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

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

\# memoise() knows about default parameters.
\text{b} \leftarrow \text{function}(n, \text{dummy}="a") \{ \text{runif}(n) \}
\n\text{memB} \leftarrow \text{memoise}(b)
\text{memB}(2)
\text{memB}(2, \text{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 caching, 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 an 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

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

cache_filesystem, 2
cache_memory, 2
cache_s3, 3
digest, 2, 3
forget, 3, 5, 6
has_cache, 4
is.memoised, 4, 5, 6
is.memoized (is.memoised), 5
memoise, 4, 5, 5, 8
memoize (memoise), 5
timeout, 6, 7