Package 'memofunc'

February 22, 2021

Type Package

Title Function Memoization

Version 1.0.2

Author Roy Wetherall <rwetherall@gmail.com>

Maintainer Roy Wetherall <rwetherall@gmail.com>

Description A simple way to memoize function results to improve performance by eliminating unnecessary computation or data retrieval activities.

Depends R (>= 3.5.0)

License GPL-3

URL https://github.com/rwetherall/memofunc, https://rwetherall.github.io/memofunc/

BugReports https://github.com/rwetherall/memofunc/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Imports digest, uuid, magrittr

Suggests testthat, devtools, roxygen2, covr

NeedsCompilation no

Repository CRAN

Date/Publication 2021-02-22 19:30:02 UTC

R topics documented:

`functionCall` *Function Call*

Description

For a given function and call, return a list of class 'functionCall' which can be hashed to provide a unique identifier for the function and parameters used for this call.

Usage

```
functionCall(f = sys.function(sys.parent()), call = sys.call(sys.parent()))
```

Arguments

`f`	function, defaults to the containing function
`call`	call, default to the containing call

Value

functionCall, a hashable form of the function call information

Examples

```
# my example function
my.function <- function (x, y) x+y

# create a new function call
my.functionCall <- functionCall(my.function, call("my.function", 10, 10))

# the function and arguments are now available
my.functionCall$f
my.functionCall$args

# using the default argument values to get the function call of the containing function
my.function2 <- function (x, y) functionCall()
my.functionCall2 <- my.function2(10, 10)

# the function and arguments are now available
my.functionCall2$f
my.functionCall2$args
```

---

hash *Hash*

Description

Hashes a value into a string.

Usage

```
hash(value)
```

Arguments

value	value to hash

Value

hashed value as a string

Examples

```
my.function <- function (x, y) x+y

# a list of values to hash
values <- list(
  "Hello world!",
  101,
  3.142,
  TRUE,
  my.function,
```

```
  (function (x, y) x+y),
  functionCall(my.function, call("my.function", 10, 10)),
  list(a=1, b=2, c="hello")
)

# hash the values in the list
(hashes <- lapply(values, hash))

# Note that functions with the same body will have the same hash
hashes[[5]] == hashes[[6]]
```

`hash.default` *Hash*

Description

Default hash function.

Usage

```
## Default S3 method:
hash(value)
```

Arguments

`value`	value to hash

Value

hashed value as a string

References

digest::digest(value)

Examples

```
my.function <- function (x, y) x+y

# a list of values to hash
values <- list(
  "Hello world!",
  101,
  3.142,
  TRUE,
  my.function,
  (function (x, y) x+y),
  functionCall(my.function, call("my.function", 10, 10)),
  list(a=1, b=2, c="hello")
)
```

```
# hash the values in the list
(hashes <- lapply(values, hash))

# Note that functions with the same body will have the same hash
hashes[[5]] == hashes[[6]]
```

`hash.function` *Hash*

Description

Hashes a function, but considering the formals and body, thus the resulting has is influenced by changes to signature and implementation.

Usage

```
## S3 method for class '`function`'
hash(value)
```

Arguments

`value`	value to hash

Value

hashed value as a string

Examples

```
my.function <- function (x, y) x+y

# a list of values to hash
values <- list(
  "Hello world!",
  101,
  3.142,
  TRUE,
  my.function,
  (function (x, y) x+y),
  functionCall(my.function, call("my.function", 10, 10)),
  list(a=1, b=2, c="hello")
)

# hash the values in the list
(hashes <- lapply(values, hash))

# Note that functions with the same body will have the same hash
hashes[[5]] == hashes[[6]]
```

`hash.functionCall` *Hash*

Description

Hashes a function call, taking into account the values provided to the function call and unprovided default values. Ensures the order the parameters are provided does not change the outcome of the hash calculation.

Usage

```
## S3 method for class 'functionCall'
hash(value)
```

Arguments

`value`	value to hash

Value

hashed value as a string

Examples

```
my.function <- function (x, y) x+y

# a list of values to hash
values <- list(
  "Hello world!",
  101,
  3.142,
  TRUE,
  my.function,
  (function (x, y) x+y),
  functionCall(my.function, call("my.function", 10, 10)),
  list(a=1, b=2, c="hello")
)

# hash the values in the list
(hashes <- lapply(values, hash))

# Note that functions with the same body will have the same hash
hashes[[5]] == hashes[[6]]
```

hash.list *Hash*

Description

Hashes a list of items, generating a single unique hash value which is based on the combination of hashed list items.

Usage

```
## S3 method for class 'list'
hash(value)
```

Arguments

value	value to hash

Value

hashed value as a string

Examples

```
my.function <- function (x, y) x+y

# a list of values to hash
values <- list(
  "Hello world!",
  101,
  3.142,
  TRUE,
  my.function,
  (function (x, y) x+y),
  functionCall(my.function, call("my.function", 10, 10)),
  list(a=1, b=2, c="hello")
)

# hash the values in the list
(hashes <- lapply(values, hash))

# Note that functions with the same body will have the same hash
hashes[[5]] == hashes[[6]]
```

is.memo *Is Memo*

Description

Checks whether the passed function is a memo function.

Usage

```
is.memo(f)
```

Arguments

f	function, memo or otherwise

Value

TRUE if memo function, FALSE otherwise

memo *Memo*

Description

Creates a memoized function, based on the provided named or anonymous function. Calls to the memoized function will be retrieved from a cache, unless it is the first time it is called.

Passing memo.force = TRUE to the memo function call will by-pass any previously cached values and execute the underlying function, storing the newly retrieved values for subsequent calls. memo.force = FALSE by default.

Passing memo.dryrun = TRUE to the memo function call will prevent the underlying function from executing and return TRUE if call isn't caches and FALSE if it is. These values are not cached as responses for the function.

Note that results are cached based on the argument values passed to the function. The order is not important since all names are resolved. So fun(a=1,b=2) will return the same cached value as fun(b=2,a=1), for example.

Functions as arguments are supported, but only the body is compared. So a named function parameter and an anonymouse function parameter with the same body, will be evaluated as identical and return the same cached value.

... is supported, but note that unless named then the order of the values is significant and will produce different cache values unless identical.

By default NULL values are not cached. Setting allow.null=TRUE when creating the memo will, however, ensure that NULL values are cached.

Usage

```
memo(f, allow.null = FALSE)
```

Arguments

f	function to memoise
allow.null	if TRUE then the memoed function will cache NULL results, otherwise it won't. FALSE by default.

Value

the memoed function

Examples

```
library(magrittr)

# a simple example function
simple.function <- function (value) {
  print("Executing!")
  value
}

# call memo function to memoise a function
simple.function.memo <- memo(simple.function)

# or like this
simple.function %<>% memo()

# or use an anon function
simple.function2 <- (function (value) value) %>% memo()

# the first time we call the memo the function will execute
simple.function(10)

# if we call the memo again with the same parameter values then
# the cached value will be returned
simple.function(10)

# calling the memo with a different set of parameter values will
# cause the function to execute
simple.function(20)

# consider a slow function which is memoised, note that we have used the allow.null argument
# so that NULL is cached when returned from a function, the default is FALSE
slow.function <- (function (value) Sys.sleep(value)) %>% memo(allow.null = TRUE)

# the first time we call the slow function it takes some time
system.time(slow.function(3))

# subsequent calls make use of the cache and are much faster
system.time(slow.function(3))
```

memo.cache *Memo Cache*

Description

Gets the cache associated with a memo function allowing further manipulation and control of the underlying values being stored.

Execution is stopped if function passed is not a valid memoed function.

Usage

```
memo.cache(f)
```

Arguments

f	memo function

Value

Cache storing values for memoed function.

memo.function *Memo Function*

Description

Gets the original function that was memoized.

Execution is stopped if function passed is not a valid memoed function.

Usage

```
memo.function(f)
```

Arguments

f	memo function

Value

Original unmemoized function.

memofunc *memofunc: A package for memoizing functions and caching data*

Description

The memofunc package provides a simple way to memoize a function to optimise execution for process or data intensive actions.

Memoization Functions

- memo - memoize a function
- is.memo - is the given function a memo
- memo.function - get a memo's origional function
- memo.cache - get a memo's cache storage

Author(s)

Roy Wetherall <rwetherall@gmail.com>

storage.clear *Clear the storage.*

Description

Clear the given storage of all keys and their values.

Usage

```
storage.clear(storage)
```

Arguments

storage	initialized storage

Value

Invisibily returns storage

Examples

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")
```

`storage.clear.default` *Clear the memory store.*

Description

Clear the given storage of all keys and their values.

Usage

```
## Default S3 method:
storage.clear(storage)
```

Arguments

storage	initialized storage

Value

Invisibily returns storage

Examples

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")
```

storage.get *Get value from a store.*

Description

Gets a value, for a given key, from the store.

If there is no coresponding value for the key, then NULL is returned.

Usage

```
storage.get(storage, key)
```

Arguments

storage	initialized storage
key	key to retrieve value for

Value

Stored value for the key, NULL otherwise.

Examples

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))
```

```
# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")
```

`storage.get.default` *Get a value from a memory store.*

Description

Gets a value, for a given key, from the store.

If there is no coresponding value for the key, then `NULL` is returned.

Usage

```
## Default S3 method:
storage.get(storage, key)
```

Arguments

`storage`	initialized storage
`key`	key to retrieve value for

Value

Stored value for the key, `NULL` otherwise.

Examples

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))
```

```
# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")
```

`storage.has` *Has key has been used to store a value?*

Description

Indicates if a given key has a associated value stored in the storage or not.

Usage

```
storage.has(storage, key)
```

Arguments

`storage`	initialized storage
`key`	key to check for stored value

Value

`TRUE` if key has an associated stored value, `FALSE` otherwise.

Examples

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")
```

---

storage.has.default *Has key has been used to store a value in a memory store?*

Description

Indicates if a given key has a associated value stored in the storage or not.

Usage

```
## Default S3 method:
storage.has(storage, key)
```

Arguments

storage	initialized storage
key	key to check for stored value

Value

TRUE if key has an associated stored value, FALSE otherwise.

Examples

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")
```

storage.init *Initialize a store.*

Description

Initlaize storage for name value pairs based on provided type.

Available types of storage include:

- `memory` - transient in-memory storage
- `file` - persistent storage, using local file storage

Additional paramters may be provided when initializing different types of storage.

See specific storage types for details.

Usage

```
storage.init(storage.type = "memory", ...)
```

Arguments

`storage.type`	storage type to initialize, defaults to `memory`
`...`	additional configuration values used by storage implementations

Value

List containing characteristics perticular to the storage implementation, including:

- `$type` - the storage type

Examples

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
```

```
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")
```

`storage.init.default` *Initialize a memory store.*

Description

Initlaize memory storage, used to hold and retrieve values in memory.

The storage type is expected to specified as `memory`.

This storage is transient.

Usage

```
## Default S3 method:
storage.init(storage.type, ...)
```

Arguments

`storage.type`	storage type to initialize, defaults to `memory`
`...`	additional configuration values used by storage implementations

Value

List containing characteristics perticular to the storage implementation, including:

- `$type` - the storage type

Examples

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")
```

`storage.set` *Set value into a store.*

Description

Stores a value for a given key.

If there is already a value stored for the key provided, then the exisiting value is overriden with the new value.

Usage

```
storage.set(storage, key, value)
```

Arguments

storage	initialized storage
key	key to store value against
value	value to store

Value

Invisbily returns storage

Examples

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
```

```
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")
```

`storage.set.default` *Set value into a memory store.*

Description

Stores a value for a given key.

If there is already a value stored for the key provided, then the exisiting value is overriden with the new value.

Usage

```
## Default S3 method:
storage.set(storage, key, value)
```

Arguments

`storage`	initialized storage
`key`	key to store value against
`value`	value to store

Value

Invisbily returns storage

Examples

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")
```

```
# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")
```

`storage.unset` *Unset a value that corresponds to a key within a store.*

Description

Unsets the value stored for a given key.

If there is no value for the key provided no action is taken.

Usage

```
storage.unset(storage, key)
```

Arguments

`storage`	initialized storage
`key`	key whose value is to be unset

Value

Invisibily returns storage

Examples

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")
```

```
# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")
```

`storage.unset.default` *Unset a value that corresponds to a key within a memory store.*

Description

Unsets the value stored for a given key.

If there is no value for the key provided no action is taken.

Usage

```
## Default S3 method:
storage.unset(storage, key)
```

Arguments

`storage`	initialized storage
`key`	key whose value is to be unset

Value

Invisibily returns storage

Examples

```
library(magrittr)

# initialize default memory storage
my.storage <- storage.init()

# set a value into storage
storage.set(my.storage, "name", "Roy Wetherall")

# .. and some more
my.storage %>%
  storage.set("age", 45) %>%
  storage.set("alive", TRUE) %>%
  storage.set("children", c("Peter", "Grace", "Lucy"))

# check a key has been set
if (storage.has(my.storage, "name")) print("I know your name!")

# .. and that a key hasn't been set
if (!storage.has(my.storage, "address")) print("I don't know where you live!")

# get some values from storage
sprintf(
  "%s is %i years old.",
  storage.get(my.storage, "name"),
  storage.get(my.storage, "age"))

# remove a value from storage
storage.unset(my.storage, "children")

# .. and show it's not there anymore
if (!storage.has(my.storage, "address")) print("I don't know who your children are!")

# clear all values from storage
storage.clear(my.storage)

# .. and everything is gone
if (!storage.has(my.storage, "name") && !storage.has(my.storage, "age")) print("I know nothing!")
```

Index