Package ‘memo’

Type Package
Title In-Memory Caching of Repeated Computations (Memoization)
Version 1.1.1
Date 2023-12-11
Author Peter Meilstrup <peter.meilstrup@gmail.com>
Maintainer Peter Meilstrup <peter.meilstrup@gmail.com>
Description A simple in-memory, LRU cache that can be wrapped
around any function to memoize it. The cache is keyed on a hash of the input data (using 'di-

gest') or on pointer equivalence.
License MIT + file LICENSE
Encoding UTF-8
Imports digest
Suggests testthat (>= 0.2), knitr, rmarkdown
Collate 'lrucache.R' 'getPointer.R' 'map.R' 'memo-description.r'
VignetteBuilder knitr
RoxygenNote 7.2.3
NeedsCompilation yes
Repository CRAN
Date/Publication 2023-12-12 00:40:02 UTC

R topics documented:

  memo-package ................................................... 2
  cache_stats ................................................. 2
  hashmap ..................................................... 3
  memo .......................................................... 4
  permanent_cache .......................................... 5
  strategies .................................................... 5

Index 7
**memo-package**

In-memory caching of repeated computations, by pointer equivalence.

**Description**

The `memo` package implements a cache that can be used to avoid repeated computations of functions. The cache lookup is based on object identity (i.e. pointer equivalence) which is suited for functions like accessors or other functions that are called repeatedly on the same object.

**Author(s)**

Peter Meilstrup

---

**cache_stats**

Report cache statistics.

**Description**

Report cache statistics.

**Usage**

`cache_stats(fn)`

**Arguments**

fn A memoized function that was created by `memo`.

**Value**

A list with labels "size", "used", "hits", "misses", "expired" counting the number of slots in the cache, the number of slots currently used, the number of times a previous result was recalled, a new result was recorded, and a result was dropped.
hashmap

A reference-valued, key-value store.

Description

[hashmap()] constructs a hashmap, which is an object that behaves like an [environment] but can key on arbitrary objects rather than just characters.

Usage

hashmap()

## S3 method for class 'hashmap'

x[...]

## S3 replacement method for class 'hashmap'

x[...] <- value

## S3 method for class 'hashmap'

x[[...]]

## S3 replacement method for class 'hashmap'

x[[...]] <- value

keys(x, ...)

values(x, ...)

to_pairs(x, ...)

from_pairs(pairs)

hasKey(x, ...)

dropKey(x, ...)

Arguments

x

a hashmap object.

... Any number of indices.

value A replacement value for '[['; for '[', a sequence of replacement values.

pairs A list of pairs, the first element is treated as key and the second as value.
Details

You can use multiple indices in a hashmap; the effect is similar to indexing on a list containing all keys.

Type is significant; for instance, float ‘1’ and integer ‘1L’ are considered distinct indices. It is also permitted to index on NULL, NA, or the empty string.

The ‘memo’ package hashmap has a performance optimization over other implementations of this concept, in that the md5 digest is memoized on scalar and pointer values. That means that if you lookup using keys that are pointer-identical to previously seen keys, it will skip computing the digest a second time. Indexing using scalar values will also bypass the md5 hash.

Value

‘hashmap()’ returns a newly constructed hashmap.
‘pairs(x)’ extracts from a hashmap a list of pairs, each pair being of the form ‘list(key=, val=)’.
‘hasKey(x)’ returns TRUE if there is a key with the same digest as ‘...’ that compares [identical()]

Author(s)

Peter Meilstrup

Description

Memoize a function.

Usage

 memo(fn, cache = lru_cache(5000), key = hybrid_key, ...) 

Arguments

 fn A function to wrap. It should be a pure function (i.e. it should not cause side effects, and should not depend on any variables that may change.) It should not be a nonstandard-evaluating function. All arguments will be forced by the wrapper.

cache A cache to use. Defaults to a new instance of lru_cache. Caches may be shared between memoized functions.

key A hashing strategy. The default "hybrid_key" first checks for pointer equivalence and then falls back to using a hash of the arguments. ‘pointer_key’ uses just pointer equivalence, and ‘digest_key’ always performs a hash.

... Further arguments passed on to key.
permanent_cache

‘basic_cache’ makes a cache that does not expire old entries. It should be used in situations where you know the number of things to remember is bounded.

Usage

permanent_cache()

lru_cache(size = 10000)

Arguments

size The maximum number of results to keep.

Value

A function f(key, value) which takes a string in the first parameter and a lazily evaluated value in the second. ‘f’ will use the string key to retrieve a value from the cache, or return the matching item from the cache, or force the second argument and return that, remembering the result on future calls.

When the number of entries in the cache exceeds size, the least recently accessed entries are removed.

strategies Strategies for caching items.

Description

The function memo accepts an argument ‘key’ which specifies the keying strategy.

Usage

digest_key(fn, cache, digest = digest::digest)

pointer_key(fn, cache)

hybrid_key(fn, cache, digest = function(x) digest::digest(x, “md5”))
Arguments

- fn: A function whose results should be cached.
- cache: A cache object.
- digest: A digest function to use.

Value

A memoized function.
Index

[.hashmap (hashmap), 3
[<-.hashmap (hashmap), 3
[[.hashmap (hashmap), 3
[[<-.hashmap (hashmap), 3
_PACKAGE (memo-package), 2

cache_stats, 2

digest_key (strategies), 5
dropKey (hashmap), 3

from_pairs (hashmap), 3

hashmap, 3
hasKey (hashmap), 3
hybrid_key, 4
hybrid_key (strategies), 5

keys (hashmap), 3

lru_cache, 4
lru_cache (permanent_cache), 5

memo, 2, 4, 5
memo-package, 2

permanent_cache, 5
pointer_key (strategies), 5

strategies, 5

to_pairs (hashmap), 3

values (hashmap), 3