Package ‘memo’

May 5, 2022

Type Package

Title In-Memory Caching of Repeated Computations (Memoization)

Version 1.0.2

Date 2022-4-28

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 'digest') or on pointer equivalence.

License MIT + file LICENSE

Imports digest

Suggests testthat (>= 0.2), knitr, rmarkdown

Collate 'lru.R' 'cache.R' 'getPointer.R' 'memo-description.r'

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-05-05 07:10:02 UTC

R topics documented:

`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.

`lru_cache` *Construct a cache with least-recently-used policy.*

Description

Construct a cache with least-recently-used policy.

Usage

```
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.

memo — *Memoize a function.*

Description

Memoize a function.

This 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. Description of memo goes here.

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. "`digest_key`". Other values include "pointer_key" and "hybrid_key".
`...`	Further arguments passed on to key.

Author(s)

Peter Meilstrup

strategies — *Strategies for caching items.*

Description

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

`digest_key` computes a key by hashing the contents of the object using the digest package. No attempt is made to avoid MD5 hash collisions.

The `pointer_key` strategy uses object identity, that is, pointer equivalence. This can be faster because hte entire object need not be hashed. However, this strategy is only useful when the function is called repeatedly on the same object rather than merely identical objects. Also be aware that the cache will hold on to the values of the arguments, to prevent them being garbage collected.

The `hybrid_key` strategy first tries to key on object identity and then falls back on computing the md5 digest. This may use two cache slots per result.

Usage

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

pointer_key(fn, cache)

hybrid_key(fn, cache, digest = digest::digest)
```

Arguments

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

Value

A memoized function.

Index