Package 'hashmap'

November 16, 2017

Type Package

Title The Faster Hash Map

Version 0.2.2

Date 2017-11-16

URL https://github.com/nathan-russell/hashmap

BugReports https://github.com/nathan-russell/hashmap/issues

Description Provides a hash table class for fast
key-value storage of atomic vector types.

License MIT + file LICENSE

LazyData Yes

ByteCompile TRUE

Imports Rcpp (>= 0.12.4)

LinkingTo Rcpp, BH

Suggests devtools, microbenchmark, testthat

Depends methods

RcppModules Hashmap

Collate 'hashmap.R' 'classes.R' 'Hashmap-class.R' 'RcppExports.R'
'clone.R' 'load_hashmap.R' 'merge.R' 'plugin.R'
'save_hashmap.R' 'zzz.R'

RoxygenNote 6.0.1

NeedsCompilation yes

Author Nathan Russell [aut, cre]

Maintainer Nathan Russell <russell.nr2012@gmail.com>

Repository CRAN

Date/Publication 2017-11-16 15:19:42 UTC

R topics documented:

clone . . . 2
hashmap . . . 3
Hashmap-class . . . 4
internal-functions . . . 7
load_hashmap . . . 8
merge . . . 9
plot . . . 10
Rcpp_Hashmap-class . . . 11
save_hashmap . . . 11

Index **13**

`clone` *Clone a Hashmap*

Description

`clone` creates a deep copy of a `Hashmap` so that modifications made to the cloned object do not affect the original object.

Usage

```
clone(x)
```

Arguments

`x`	an object created by a call to `hashmap`.

Details

Since the actual cloning is done in C++, `y <- clone(x)` should be much more efficient than `y <- hashmap(x$keys(), x$values())`.

Value

a `Hashmap` identical to the input object.

See Also

`hashmap`

Examples

```
x <- hashmap(letters[1:5], 1:5)

## shallow copy
y <- x
y["a"]] <- 999

## original is affected
x["a"]] == 999

z <- clone(x)
z["c"]] <- 888

## original not affected
x["c"]] == 888
```

---

hashmap *Atomic vector hash map*

Description

Create a new Hashmap instance

Usage

```
hashmap(keys, values, ...)
```

Arguments

keys	an atomic vector representing lookup keys
values	an atomic vector of values associated with keys in a pair-wise manner
...	other arguments passed to new when constructing the Hashmap instance

Details

The following atomic vector types are currently supported for keys:

- integer
- numeric
- character
- Date
- POSIXct

The following atomic vector types are currently supported for values:

- logical

- integer
- numeric
- character
- complex
- Date
- POSIXct

Value

a Hashmap object

See Also

Hashmap-class for a more detailed discussion of available methods

Examples

```
x <- replicate(10e3,
    paste0(sample(letters, 12, TRUE),
           collapse = "")
)
y <- rnorm(length(x))
z <- sample(x, 100)

H <- hashmap(x, y)

all.equal(y[match(z, x)], H[[z]])

## Not run:
microbenchmark::microbenchmark(
    "R" = y[match(z, x)],
    "H" = H[[z]],
    times = 500L
)

## End(Not run)
```

Hashmap-class *Internal hash map class*

Description

A C++ class providing hash map functionality for atomic vectors

Details

A Hashmap object (H) resulting from a call to hashmap(keys, values) provides the following methods accessible via $method_name:

- keys(): returns the keys of H.
- values(): returns the values of H.
- cache_keys(): caches an internal vector with the hash table's current keys resulting in very low overhead calls to keys(). For larger hash tables this has a significant effect. However, any calls to modifying functions (clear, insert, etc.) will invalidate the cached state.
- cache_values(): caches an internal vector with the hash table's current values resulting in very low overhead calls to values(). For larger hash tables this has a significant effect. However, any calls to modifying functions (clear, insert, etc.) will invalidate the cached state.
- keys_cached(): returns TRUE if the hash table's keys are currently cached, and FALSE otherwise.
- values_cached(): returns TRUE if the hash table's values are currently cached, and FALSE otherwise.
- erase(remove_keys): deletes entries for elements that exist in the hash table, and ignores elements that do not.
- clear(): deletes all keys and values from H.
- data(): returns a named vector of values using the keys of H as names.
- empty(): returns TRUE if H is empty (e.g. immediately following a call to clear), else returns FALSE.
- find(lookup_keys): returns the values associated with lookup_keys for existing key elements, and NA otherwise.
- has_key(lookup_key): returns TRUE if lookup_key exists as a key in H and FALSE if it does not.
- has_keys(lookup_keys): vectorized equivalent of has_key.
- rehash(n_buckets): for the internal hash table, sets the number of buckets to at least n and the load factor to less than the max load factor.
- bucket_count(): returns the current number of buckets in the internal hash table.
- hash_value(keys): compute hash values for the vector keys using the hash table's internal hash function. Note that keys need not exist in the hash table, but it must have the same type as the hash table's keys. This can be useful for investigating the efficacy of the object's hash function.
- renew(new_keys, new_values): deletes current keys and values, and reinitialize H with new_keys and new_values, where new_keys and new_values are allowed to be different SEXP types than the original keys and values.
- insert(more_keys, more_values): adds more key-value pairs to H, where existing key elements (intersect(H$keys(), more_keys)) will be updated with the corresponding elements in more_values, and non-existing key elements setdiff(H$keys(), more_keys) will be inserted with the corresponding elements in more_values.
- size(): returns the size (number of key-value pairs) of (held by) H.

Additionally, the following two convenience methods which do not require the use of `$`:

- `` `[[` ``: equivalent to `find(lookup_keys)`.
- `` `[[<-` ``: equivalent to `insert(more_keys, more_values)`.

Examples

```
x <- replicate(10e3,
    paste0(sample(letters, 12, TRUE),
           collapse = "")
)
y <- rnorm(length(x))
z <- sample(x, 100)

H <- hashmap(x, y)

H$empty()       #[1] FALSE
H$size()        #[1] 10000

## necessarily
any(duplicated(H$keys()))        #[1] FALSE

all.equal(H[[z]], H$find(z))  #[1] TRUE

## hash map ordering is random
all.equal(
    sort(H[[x]]),
    sort(H$values()))  #[1] TRUE

## a named vector
head(H$data())

## redundant, but TRUE
all.equal(
    H[[names(head(H$data()))]],
    unname(head(H$data())))

## setting values
H2 <- hashmap(H$keys(), H$values())

all.equal(
    sort(H[[H2$keys()]]),
    sort(H2[[H$keys()]]))   #[1] TRUE

H$insert("A", round(pi, 5))

H2[["A"]] <- round(pi, 5)

## still true
all.equal(
    sort(H[[H2$keys()]]),
    sort(H2[[H$keys()]]))
```

```
## changing SEXPTYPE of key or value must be explicit
H3 <- hashmap(c("A", "B", "C"), c(1, 2, 3))

H3$size()     #[1] 3

H3$clear()
H3$size()     #[1] 0

## not allowed
class(try(H3[["D"]] <- "text", silent = TRUE)) #[1] "try-error"

## okay
H3$renew("D", "text")
H3$size()     #[1] 1
```

`internal-functions` *Hashmap internal functions*

Description

Hashmap internal functions

Usage

```
.left_outer_join_impl(x, y)

.right_outer_join_impl(x, y)

.inner_join_impl(x, y)

.full_outer_join_impl(x, y)
```

Arguments

`x`	an external pointer to a `HashMap`
`y`	an external pointer to a `HashMap`

Details

These functions are intended for internal use only; do not call them directly.

load_hashmap *Load Hashmaps*

Description

load_hashmap reads a file created by a call to save_hashmap and returns a Hashmap object.

Usage

```
load_hashmap(file)
```

Arguments

file	the name of a file previously created by a call to save_hashmap.

Details

The object returned will contain all of the same key-value pairs that were present in the original Hashmap at the time save_hashmap was called, but they are not guaranteed to be in the same order, due to rehashing.

Value

A Hashmap object on success; an error on failure.

See Also

save_hashmap

Examples

```
H <- hashmap(sample(letters[1:10]), sample(1:10))
tf <- tempfile()

save_hashmap(H, tf)

H2 <- load_hashmap(tf)
all.equal(
    sort(H$find(H2$keys())),
    sort(H2$values())
)

all.equal(
    H$data.frame(),
    readRDS(tf)
)
```

merge *Merge two Hashmaps*

Description

merge method for Hashmap class

Usage

```
## S3 method for class 'Rcpp_Hashmap'
merge(x, y, type = c("inner", "left", "right", "full"),
  ...)
```

Arguments

x	an object created by a call to hashmap.
y	an object created by a call to hashmap.
type	a character string specifying the type of join, with partial argument matching (abbreviation) supported.
...	not used.

Details

Valid arguments for type are:

- "inner": similar to all = FALSE in base::merge
- "left": similar to all.x = TRUE in base::merge
- "right": similar to all.y = TRUE in base::merge
- "full": similar to all = TRUE in base::merge

The default value for type is "inner".

Value

a data.frame.

Examples

```
hx <- hashmap(LETTERS[1:5], 1:5)
hy <- hashmap(LETTERS[4:8], 4:8)

## inner join
merge(hx, hy)

merge(
    hx$data.frame(),
    hy$data.frame(),
```

```
    by = "Keys",
    sort = FALSE
)

## left join
merge(hx, hy, "left")

merge(
    hx$data.frame(),
    hy$data.frame(),
    by = "Keys",
    all.x = TRUE,
    sort = FALSE
)

## right join
merge(hx, hy, "right")

merge(
    hx$data.frame(),
    hy$data.frame(),
    by = "Keys",
    all.y = TRUE,
    sort = FALSE
)

## full outer join
merge(hx, hy, "full")

merge(
    hx$data.frame(),
    hy$data.frame(),
    by = "Keys",
    all = TRUE,
    sort = FALSE
)
```

`plot` *Plot method for Hashmap class*

Description

Plot method for Hashmap class

Usage

```
## S3 method for class 'Rcpp_Hashmap'
plot(x, ...)
```

Arguments

x	an object created by a call to hashmap
...	arguments passed to plot

Details

A convenience function which simply calls plot using x$keys() and x$values() as plotting co-ordinates.

Examples

```
x <- hashmap(1:20, rnorm(20))
plot(x)
plot(x, type = 'p', pch = 20, col = 'red')

y <- hashmap(Sys.Date() + 1:20, rnorm(20))
plot(y, type = 'h', col = 'blue', lwd = 3)
```

Rcpp_Hashmap-class *Hashmap internal class*

Description

Hashmap internal class

---

save_hashmap *Save Hashmaps*

Description

save_hashmap writes a Hashmap's data to the specified file, which can be passed to load_hashmap at a later point in time to recreate the object.

Usage

```
save_hashmap(x, file, overwrite = TRUE, compress = FALSE)
```

Arguments

x	an object created by a call to hashmap.
file	a filename where the object's data will be saved.
overwrite	if TRUE (default) and file exists, it will be overwritten. If FALSE and file exists, an error is thrown.
compress	a logical value or the type of file compression to use; defaults to FALSE for better performance. See ?saveRDS for details.

Details

Saving is done by calling `base::saveRDS` on the object's `data.frame` representation, `x$data.frame()`. Attempting to save an empty `Hashmap` results in an error.

Value

Nothing on success; an error on failure.

See Also

`load_hashmap`, `saveRDS`

Examples

```
H <- hashmap(sample(letters[1:10]), sample(1:10))
tf <- tempfile()

save_hashmap(H, tf)

inherits(
    try(save_hashmap(H, tf, FALSE), silent = TRUE),
    "try-error"
)

H$insert("zzzzz", 123L)
save_hashmap(H, tf)

load_hashmap(tf)
```

Index

.full_outer_join_impl
(internal-functions), 7
.inner_join_impl (internal-functions), 7
.left_outer_join_impl
(internal-functions), 7
.right_outer_join_impl
(internal-functions), 7

clone, 2

Hashmap (Hashmap-class), 4
hashmap, *2*, 3
Hashmap-class, 4

internal-functions, 7

load_hashmap, 8, *11*, *12*

merge, 9

plot, 10

Rcpp_Hashmap (Rcpp_Hashmap-class), 11
Rcpp_Hashmap-class, 11

save_hashmap, *8*, 11
saveRDS, *12*