Package 'filehash'

February 9, 2023

Version 2.4-5

Depends R (>= 3.0.0)

Imports digest, methods

Collate filehash.R filehash-DB1.R filehash-RDS.R coerce.R dump.R hash.R queue.R stack.R zzz.R

Title Simple Key-Value Database

Author Roger D. Peng <rdpeng@jhu.edu>

Maintainer Roger D. Peng <rdpeng@jhu.edu>

Description Implements a simple key-value style database where character string keys are associated with data values that are stored on the disk. A simple interface is provided for inserting, retrieving, and deleting data from the database. Utilities are provided that allow 'filehash' databases to be treated much like environments and lists are already used in R. These utilities are provided to encourage interactive and exploratory analysis on large datasets. Three different file formats for representing the database are currently available and new formats can easily be incorporated by third parties for use in the 'filehash' framework.

License GPL (>= 2)

URL https://github.com/rdpeng/filehash

RoxygenNote 7.2.3

Encoding UTF-8

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-02-09 18:40:02 UTC

R topics documented:

`coerceDB1list` *Coerce a filehash database*

Description

Coerce a filehashDB1 database to a list object

Arguments

`from`	a filehashDB1 database object

`coerceDB1RDS` *Coerce a filehash database*

Description

Coerce a filehashDB1 database to filehashRDS format

Arguments

`from`	a filehashDB1 database object

`coercelist` *Coerce a filehash database*

Description

Coerce a filehash database to a list object

Arguments

`from`	a filehash database object

dbLoad *Load a Database*

Description

Load entire database into an environment

Usage

```
dbLoad(db, ...)

## S4 method for signature 'filehash'
dbLoad(db, env = parent.frame(2), keys = NULL, ...)

dbLazyLoad(db, ...)

## S4 method for signature 'filehash'
dbLazyLoad(db, env = parent.frame(2), keys = NULL, ...)

db2env(db)
```

Arguments

db	filehash database object
...	arguments passed to other methods
env	environment into which objects should be loaded
keys	specific keys to be loaded (if NULL then all keys are loaded)

Details

dbLoad loads objects in the database directly into the environment specified, like load does except with active bindings. dbLoad takes a second argument env, which is an environment, and the default for env is parent.frame().

The use of makeActiveBinding in db2env and dbLoad allows for potentially large databases to, at least conceptually, be used in R, as long as you don't need simultaneous access to all of the elements in the database.

dbLazyLoad loads objects in the database directly into the environment specified, like load does except with promises. dbLazyLoad takes a second argument env, which is an environment, and the default for env is parent.frame().

With dbLazyLoad database objects are "lazy-loaded" into the environment. Promises to load the objects are created in the environment specified by env. Upon first access, those objects are copied into the environment and will from then on reside in memory. Changes to the database will not be reflected in the object residing in the environment after first access. Conversely, changes to the object in the environment will not be reflected in the database. This type of loading is useful for read-only databases.

db2env loads the entire database db into an environment via calls to makeActiveBinding. Therefore, the data themselves are not stored in the environment, but a function pointing to the data in the database is stored. When an element of the environment is accessed, the function is called to retrieve the data from the database. If the data in the database is changed, the changes will be reflected in the environment.

Value

dbLoad, dbLazyLoad: a character vector is returned (invisibly) containing the keys associated with the values loaded into the environment.

db2env: environment containing database keys

Methods (by class)

- dbLoad(filehash): Method for filehash databases
- dbLazyLoad(filehash): Method for filehash databases

Functions

- dbLazyLoad(): Lazy load a filehash database
- db2env(): Load active bindings into an environment and return the environment

See Also

dbLoad, dbLazyLoad

---

dumpEnv *Dump Environment*

Description

Dump an enviroment to a filehash database

Usage

```
dumpEnv(env, dbName)

dumpImage(dbName = "Rworkspace", type = NULL)

dumpObjects(
  ...,
  list = character(0),
  dbName,
  type = NULL,
  envir = parent.frame()
)
```

```
dumpDF(data, dbName = NULL, type = NULL)

dumpList(data, dbName = NULL, type = NULL)
```

Arguments

env	an environment
dbName	character, name of the filehash database
type	type of filehash database to create
...	R objects to be dumped to a filehash database
list,	character vector of object names to be dumped
envir	environment from which objects are dumped
data	a data frame

Details

The dumpEnv function takes an environment and stores each element of the environment in a filehash database. Objects dumped to a database can later be loaded via dbLoad or can be accessed with dbFetch, dbList, etc. Alternatively, the with method can be used to evaluate code in the context of a database. If a database with name dbName already exists, objects will be inserted into the existing database (and values for already-existing keys will be overwritten).

dumpDF is different in that each variable in the data frame is stored as a separate object in the database. So each variable can be read from the database separately rather than having to load the entire data frame into memory. dumpList works in a simlar way.

Value

An object of class "filehash" is returned and a database is created.

Functions

- dumpImage(): Dump the Global Environment (analogous to save.image)
- dumpObjects(): Dump named objects to a filehash database (analogous to save)
- dumpDF(): Dump data frame columns to a filehash database
- dumpList(): Dump elements of a list to a filehash database

filehash-class *Filehash Class*

Description

These functions form the interface for a simple file-based key-value database (i.e. hash table).

Usage

```
## S4 method for signature 'filehash'
show(object)

## S4 method for signature 'ANY'
dbCreate(db, type = NULL, ...)

## S4 method for signature 'ANY'
dbInit(db, type = NULL, ...)

## S4 method for signature 'filehash'
names(x)

## S4 method for signature 'filehash'
length(x)

## S4 method for signature 'filehash'
with(data, expr, ...)

## S4 method for signature 'filehash'
lapply(X, FUN, ..., keep.names = TRUE)

dbMultiFetch(db, key, ...)

dbInsert(db, key, value, ...)

dbFetch(db, key, ...)

dbExists(db, key, ...)

dbList(db, ...)

dbDelete(db, key, ...)

dbReorganize(db, ...)

dbUnlink(db, ...)

## S4 method for signature 'filehash,character,missing'
```

```
x[[i, j]]

## S4 method for signature 'filehash'
x$name

## S4 replacement method for signature 'filehash,character,missing'
x[[i, j]] <- value

## S4 replacement method for signature 'filehash'
x$name <- value

## S4 method for signature 'filehash,character,missing,missing'
x[i, j, drop]
```

Arguments

object	a filehash object
db	a filehash object
type	filehash database type
...	arguments passed to other methods
x	a filehash object
data	a filehash object
expr	an R expression to be evaluated
X	a filehash object
FUN	a function to be applied
keep.names	Should the key names be returned in the resulting list?
key	a character vector indicating a key (or keys) to retreive
value	an R object
i	a character index
j	not used
name	the name of the element in the filehash database
drop	should dimensions be dropped? (not used)

Details

Objects can be created by calls of the form `new("filehash", ...)`.

Methods (by generic)

- `show(filehash)`: Print a filehash object
- `dbCreate(ANY)`: Create a filehash database
- `dbInit(ANY)`: Initialize an existing filehash database
- `names(filehash)`: Return the keys stored in a filehash database

- `length(filehash)`: Return the number of objects in a filehash database
- `with(filehash)`: Use a filehash database as an evaluation environment
- `lapply(filehash)`: Apply a function over the elements of a filehash database
- `x[[i`: Extract elements of a filehash database using character names
- `$`: Extract elements of a filehash database using character names
- `` `[[`(x = filehash, i = character, j = missing) <- value ``: Replace elements of a filehash database
- `` `$`(filehash) <- value ``: Replace elements of a filehash database
- `x[i`: Retrieve multiple elements of a filehash database

Functions

- `dbMultiFetch()`: Retrieve values associated with multiple keys (a list of those values is returned).
- `dbInsert()`: Insert a key-value pair into the database. If that key already exists, its associated value is overwritten. For `"RDS"` type databases, there is a `safe` option (defaults to `TRUE`) which allows the user to insert objects somewhat more safely (objects should not be lost in the event of an interrupt).
- `dbFetch()`: Retrieve the value associated with a given key.
- `dbExists()`: Check to see if a key exists.
- `dbList()`: List all keys in the database.
- `dbDelete()`: The `dbDelete` function is for deleting elements, but for the `"DB1"` format all it does is remove the key from the lookup table. The actual data are still in the database (but inaccessible). If you reinsert data for the same key, the new data are simply appended on to the end of the file. Therefore, it's possible to have multiple copies of data lying around after a while, potentially making the database file big. The `"RDS"` format does not have this problem.
- `dbReorganize()`: The `dbReorganize` function is there for the purpose of rewriting the database to remove all of the stale entries. Basically, this function creates a new copy of the database and then overwrites the old copy. This function has not been tested extensively and so should be considered *experimental*. `dbReorganize` is not needed when using the `"RDS"` format.
- `dbUnlink()`: Delete an entire database from the disk.

Slots

`name` Object of class `"character"`, name of the database.

`filehashDB1-class` *Filehash DB1 Class*

Description

An implementation of filehash databases using a single large file

Usage

```
## S4 method for signature 'filehashDB1,character'
dbInsert(db, key, value, ...)

## S4 method for signature 'filehashDB1,character'
dbFetch(db, key, ...)

## S4 method for signature 'filehashDB1,character'
dbMultiFetch(db, key, ...)

## S4 method for signature 'filehashDB1,character'
dbExists(db, key, ...)

## S4 method for signature 'filehashDB1'
dbList(db, ...)

## S4 method for signature 'filehashDB1,character'
dbDelete(db, key, ...)

## S4 method for signature 'filehashDB1'
dbUnlink(db, ...)

## S4 method for signature 'filehashDB1'
dbReorganize(db, ...)
```

Arguments

`db`	a filehashDB1 object
`key`	character, the name of an R object in the database
`value`	an R object
`...`	arguments passed to other methods

Details

For `dbMultiFetch`, `key` is a character vector of keys.

Methods (by generic)

- `dbInsert(db = filehashDB1, key = character)`: Insert an R object into a filehashDB1 database
- `dbFetch(db = filehashDB1, key = character)`: Retrieve an object from a filehash DB1 database
- `dbMultiFetch(db = filehashDB1, key = character)`: Retrieve multiple objects from a filehash DB1 database
- `dbExists(db = filehashDB1, key = character)`: Determine if a key exists in a filehash DB1 database
- `dbList(filehashDB1)`: Return a character vector containing all keys in a database
- `dbDelete(db = filehashDB1, key = character)`: Delete a key and it's corresponding object from a filehashDB1 database
- `dbUnlink(filehashDB1)`: Delete an entire filehashDB1 database
- `dbReorganize(filehashDB1)`: Reorganize and compactify a filehahsDB1 database

Slots

`datafile` full path to the database file (filehashDB1 only)

`meta` list containing an environment for database metadata (filehashDB1 only)

`filehashFormats` *List and register filehash formats*

Description

List and register filehash backend database formats.

Usage

```
filehashFormats(...)
```

Arguments

`...`	list of functions for registering a new database format

Details

`filehashFormats` can be used to register new filehash backend database formats. `filehashFormats` called with no arguments lists information on available formats

Value

A list containing information on the available filehash formats

filehashOption *Set Filehash Options*

Description

Set global filehash options

Usage

```
filehashOption(...)
```

Arguments

...	name-value pairs for options

Details

Currently, the only option that can be set is the default database type (`defaultType`) which can be "DB1", "RDS" or "DB".

Value

`filehashOptions` returns a list of current settings for all options.

filehashRDS-class *Filehash RDS Class*

Description

An implementation of filehash databases using diretories and separate files

Usage

```
## S4 method for signature 'filehashRDS,character'
dbInsert(db, key, value, safe = TRUE, ...)

## S4 method for signature 'filehashRDS,character'
dbFetch(db, key, ...)

## S4 method for signature 'filehashRDS,character'
dbMultiFetch(db, key, ...)

## S4 method for signature 'filehashRDS,character'
dbExists(db, key, ...)

## S4 method for signature 'filehashRDS'
```

```
dbList(db, ...)

## S4 method for signature 'filehashRDS,character'
dbDelete(db, key, ...)

## S4 method for signature 'filehashRDS'
dbUnlink(db, ...)
```

Arguments

db	a filehashRDS object
key	character, the name of an R object
value	an R object
safe	Should the operation be done safely?
...	arguments passed to other methods

Details

When safe = TRUE in dbInsert, objects are written to a temp file before replacing any existing objects. This way, if the operation is interrupted, the original data are not corrupted.

For dbMultiFetch, key is a character vector of keys.

Methods (by generic)

- dbInsert(db = filehashRDS, key = character): Insert an R object into a filehashRDS database
- dbFetch(db = filehashRDS, key = character): Retrieve a value from a filehashRDS database
- dbMultiFetch(db = filehashRDS, key = character): Retrieve multiple objects from a filehashRDS database
- dbExists(db = filehashRDS, key = character): Determine if a key exists in a filehashRDS database
- dbList(filehashRDS): Return a character vector of all key stored in a database
- dbDelete(db = filehashRDS, key = character): Delete a key and its corresponding object from a filehashRDS database
- dbUnlink(filehashRDS): Delete an entire filehashRDS database

Slots

dir Directory where files are stored (filehashRDS only)

queue-class *A Queue Class*

Description

A queue implementation using a `filehash` database

Usage

```
createQ(filename)

initQ(filename)

pop(db, ...)

push(db, val, ...)

isEmpty(db, ...)

top(db, ...)

## S4 method for signature 'queue'
show(object)

## S4 method for signature 'queue'
push(db, val, ...)

## S4 method for signature 'queue'
isEmpty(db)

## S4 method for signature 'queue'
top(db, ...)

## S4 method for signature 'queue'
pop(db, ...)
```

Arguments

`filename`	name of queue file
`db`	a queue object
`...`	arguments passed to other methods
`val`	an R object to be added to the tail queue
`object`	a queue object

Details

Objects can be created by calls of the form `new("queue", ...)` or by calling `createQ`. Existing queues can be initialized with `initQ`.

Value

`createQ` and `initQ` return a queue object

Methods (by generic)

- `show(queue)`: Print a queue object
- `push(queue)`: adds an element to the tail ("bottom") of the queue
- `isEmpty(queue)`: returns `TRUE`/`FALSE` depending on whether there are elements in the queue.
- `top(queue)`: returns the value of the "top" (i.e. head) of the queue; an error is signaled if the queue is empty
- `pop(queue)`: returns the value of the "top" (i.e. head) of the queue and subsequently removes that element from the queue; an error is signaled if the queue is empty

Functions

- `createQ()`: Create a file-based queue object
- `initQ()`: Intialize an existing queue object
- `pop()`: Return (and remove) the top element of a queue
- `push()`: Push an R object on to the tail of a queue
- `isEmpty()`: Check if a queue is empty or not
- `top()`: Return the top of the queue

Slots

`queue` Object of class `"filehashDB1"`

`name` Object of class `"character"`: the name of the queue (default is the file name in which the queue data are stored)

---

registerFormatDB *Register Database Format*

Description

Register Database Format

Usage

```
registerFormatDB(name, funlist)
```

Arguments

name	character, name of database format
funlist	list of functions for creating and initializing a database format

---

stack-class *Stack Class*

Description

A stack implementation using a filehash database

Usage

```
## S4 method for signature 'stack'
show(object)

createS(filename)

initS(filename)

## S4 method for signature 'stack'
push(db, val, ...)

mpush(db, vals, ...)

## S4 method for signature 'stack'
mpush(db, vals, ...)

## S4 method for signature 'stack'
isEmpty(db, ...)

## S4 method for signature 'stack'
top(db, ...)

## S4 method for signature 'stack'
pop(db, ...)
```

Arguments

object	a stack object
filename	name of file where stack is stored
db	a stack object
val	an R object to be added to the stack
...	arguments passed to other methods
vals	a list of R objects

Details

Objects can be created by calls of the form `new("stack", ...)` or by calling `createS`. Existing queues can be initialized with `initS`.

Value

a stack object

Methods (by generic)

- `show(stack)`: Print a stack object.
- `push(stack)`: Push an object on to the stack
- `mpush(stack)`: Push a list of R objects on to the stack
- `isEmpty(stack)`: Indicate whether the stack is empty or not
- `top(stack)`: Return the top element of the stack
- `pop(stack)`: Return the top element of the stack and remove that element from the stack

Functions

- `createS()`: Create a filehash Stack
- `initS()`: Initialize and existing filehash stack
- `mpush()`: Push multiple R objects on to a stack

Slots

`stack` Object of class `"filehashDB1"`

`name` Object of class `"character"`: the name of the stack (default is the file name in which the stack data are stored)

Index