Package 'SharedObject'

December 3, 2020

Type Package

Title Sharing R objects across multiple R processes without memory duplication

Version 1.4.0

Date 2019-6-10

Description This package is developed for facilitating parallel computing in R.
It is capable to create an R object in the shared memory space and share the data across multiple R processes.
It avoids the overhead of memory dulplication and data transfer, which make sharing big data object across many clusters possible.

License GPL-3

LinkingTo BH, Rcpp

Depends R (>= 3.6.0)

Imports Rcpp, methods, stats, BiocGenerics

biocViews Infrastructure

BugReports https://github.com/Jiefei-Wang/SharedObject/issues

Suggests testthat, parallel, knitr, rmarkdown, BiocStyle

RoxygenNote 7.1.1

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

SystemRequirements GNU make, C++11

Encoding UTF-8

git_url https://git.bioconductor.org/packages/SharedObject

git_branch RELEASE_3_12

git_last_commit ceb0939

git_last_commit_date 2020-10-30

Date/Publication 2020-12-02

Author Jiefei Wang [aut, cre]

Maintainer Jiefei Wang <jwang96@buffalo.edu>

R topics documented:

`getLastIndex` *Functions to manipulate shared memory*

Description

These functions are for package developers only, they can allocate, open, close and destroy shared memory without touching C++ code. Normal users should not use these functions unless dealing with memory leaking

Usage

```
getLastIndex()

allocateSharedMemory(size)

allocateNamedSharedMemory(name, size)

mapSharedMemory(x)

unmapSharedMemory(x)

freeSharedMemory(x)

hasSharedMemory(x)

getSharedMemorySize(x)

initialSharedObjectPackageData()

releaseSharedObjectPackageData()
```

Arguments

`size`	numeric(1), the size of the shared memory that you want to allocate
`name`	character(1), a single name that names the shared memory
`x`	integer(1) or character(1), an ID or a name that is used to find the shared memory. If x is a character with pure number, it will be treated as an ID.

Details

Quick explanation

`getLastIndex`: the ID of the last created shared memory.

`allocateSharedMemory`: allocate a shared memory of a given size, the memory ID is returned by the function

`allocateNamedSharedMemory`: allocate a shared memory of a given size, the memory can be found by the name that is passed to the function.

`mapSharedMemory`: map the shared memory to the current process memory space

`unmapSharedMemory`: unmap the shared memory(without destroying it)

`freeSharedMemory`: destroy the shared memory. This function will only unmap the shared memory on Windows.

`hasSharedMemory`: whether the memory exist?

`getSharedMemorySize`: get the actual size of the shared memory, it may be larger than the size you required.

Details

Creating and using shared memory involves three steps: allocating, mapping, and destroying the shared memory. There are two types of naming scheme that you can use to find the shared memory: an integer ID or a character name. They are determined in the first creation step.

The shared memory can be created by `allocateSharedMemory` or `allocateNamedSharedMemory`. The function `allocateSharedMemory` will return the ID of the shared memory. After creating the shared memory, it can be mapped to the current process by `mapSharedMemory`. The return value is an external pointer to the shared memory. Once the shared memory is no longer needed, it can be destroyed by `freeSharedMemory`. There is no need to unmap the shared memory unless you intentionally want to do so.

Value

`getLastIndex`: An interger ID served as a hint of the last created shared memory ID.

`allocateSharedMemory`: an integer ID that can be used to find the shared memory

`allocateNamedSharedMemory`: no return value

`mapSharedMemory`: An external pointer to the shared memory

`unmapSharedMemory`: Logical value indicating whether the operation is success.

`freeSharedMemory`: Logical value indicating whether the operation is success.

`hasSharedMemory`: Logical value indicating whether the shared memory exist

`getSharedMemorySize`: A numeric value

See Also

`listSharedObject`

Examples

```
size <- 10L
## unnamed shared memory
id <- allocateSharedMemory(size)
hasSharedMemory(id)
ptr <- mapSharedMemory(id)
```

```
ptr
getSharedMemorySize(id)
freeSharedMemory(id)
hasSharedMemory(id)

## named shared memory
name <- "SharedObjectExample"
if(!hasSharedMemory(name)){
    allocateNamedSharedMemory(name,size)
    hasSharedMemory(name)
    ptr <- mapSharedMemory(name)
    ptr
    getSharedMemorySize(name)
    freeSharedMemory(name)
    hasSharedMemory(name)
}
```

`getSharedObjectProperty` *Get/Set the properties of the shared object.*

Description

Get/Set the properties of the shared object. The available properties are `dataId`, `length`, `totalSize`, `dataType`, `ownData`, `copyOnWrite`, `sharedSubset`, `sharedCopy`.

Usage

```
getSharedObjectProperty(x, property = NULL, ...)

setSharedObjectProperty(x, property, value, ...)

## S4 method for signature 'ANY,characterOrNULLOrMissing'
getSharedObjectProperty(x, property = NULL, ...)

## S4 method for signature 'list,characterOrNULLOrMissing'
getSharedObjectProperty(x, property = NULL, ...)

## S4 method for signature 'ANY,characterOrNULLOrMissing'
setSharedObjectProperty(x, property, value, ...)

## S4 method for signature 'list,characterOrNULLOrMissing'
setSharedObjectProperty(x, property, value, ...)

getCopyOnWrite(x)

getSharedSubset(x)

getSharedCopy(x)

setCopyOnWrite(x, value)
```

```
setSharedSubset(x, value)

setSharedCopy(x, value)
```

Arguments

x	A shared object
property	A character vector. The name of the property(s), if the argument is missing or the value is NULL, it represents all properties.
...	Not used
value	The new value of the property, if the length of value does not match the length of the property, the argument value will be repeated to match the length.

Value

get: The property(s) of a shared object

set: No return value

Examples

```
x = share(1:20)

## Check the default values
getSharedObjectProperty(x, NULL)
getCopyOnWrite(x)
getSharedSubset(x)
getSharedCopy(x)

## Set the values
setCopyOnWrite(x, FALSE)
setSharedSubset(x, FALSE)
setSharedCopy(x, TRUE)

## Check the values again
getSharedObjectProperty(x, NULL)
getCopyOnWrite(x)
getSharedSubset(x)
getSharedCopy(x)
```

---

is.altrep *Whether an object is an ALTREP object*

Description

Whether an object is an ALTREP object

Usage

```
is.altrep(x)
```

Arguments

x	an R object

Value

A logical value

Examples

```
x <- share(runif(10))
is.altrep(x)
```

`is.shared` *Test whether the object is a shared object*

Description

Test whether the object is a shared object

Usage

```
is.shared(x, ..., depth = 0, showAttributes = FALSE)

## S4 method for signature 'ANY'
is.shared(x, ..., depth = 0, showAttributes = FALSE)
```

Arguments

`x`	An R object
`...`	For generalization purpose only
`depth`	Whether to recursively check the element of `x` if `x` is a container (e.g. `list` or `environment`), see details
`showAttributes`	Whether to check the attributes of `x`.

Details

When `depth=0`, the `is.shared` function return a single logical value indicating whether `x` contains any shared objects. When `depth>0` and `x` is a container(e.g. `list`), the function will recursively check each element of `x` and return a list with each elements corresponding to the elements in `x`. The depth of the checking procedure is determined by the `depth` parameter.

if `showAttributes = TRUE`, the attributes of the object will also be checked. The result can be find in `sharedAttributes` attribute. Note that `showAttributes` has no effect on an S4 object for the attributes of an S4 object are used to store the slots and should not be treated as the attributes of an object.

Value

a single logical value of a list.

Examples

```
x1 <- share(1:10)
is.shared(x1)

x2 <- share(list(a=list(b=1, d=2)))
is.shared(x2, depth=0)
is.shared(x2, depth=1)
is.shared(x2, depth=2)
```

`listSharedObject` *Get the shared object usage report*

Description

Get the shared object usage report. The size is the real memory size that a system allocates for the shared object, so it might be larger than the object size. The size unit is byte.

Usage

```
listSharedObject(end = NULL, start = NULL, includeCharId = FALSE)
```

Arguments

`end`	the end value of the ID. The default is `NULL`. See details.
`start`	the start value of the ID. The default is `NULL`. See details.
`includeCharId`	Whether including the shared objects named by a character ID, it only works on some linux systems. See details and `?allocateNamedSharedMemory` for more information. The default is `FALSE`.

Details

The parameter `start` and `end` specify the range of the ID. If not specified, all IDs will be listed.

On Ubuntu or some other linux systems, the shared objects can be found in the folder /dev/shm. The function can find all shared objects if the folder exists.

On Windows, since there is no easy way to find all shared objects. the function will guess the range of the shared object IDs and search all IDs within the range. Therefore, if there are too many shared objects(over 4 billions) ,the object id can be out of the searching range and the result may not be complete. Furthermore, there will be no named shared object in the returned list.

Value

A data.frame object with shared object id and size

See Also

`getLastIndex`, `allocateSharedMemory`, `allocateNamedSharedMemory`, `mapSharedMemory`, `unmapSharedMemory`, `freeSharedMemory`, `hasSharedMemory`, `getSharedMemorySize`

Examples

```
## Automatically determine the search range
listSharedObject()

## specify the search range
listSharedObject(start = 10, end = 20)

## Search from 0 to 20
listSharedObject(20)
```

`pkgconfig` *Find path of the shared memory header file*

Description

This function will return the path of the shared memory header or the flags that are used to compile the package for the developers who want to use C++ level implementation of the `SharedObject` package

Usage

```
pkgconfig(x)
```

Arguments

`x`	Character, "PKG_LIBS" or "PKG_CPPFLAGS"

Value

path to the header or compiler flags

Examples

```
SharedObject:::pkgconfig("PKG_LIBS")
SharedObject:::pkgconfig("PKG_CPPFLAGS")
```

`setSharedObjectOptions` *Get or set the global options for the SharedObject package*

Description

Get or set the global options for the SharedObject package

Usage

```
setSharedObjectOptions(...)

getSharedObjectOptions(...)
```

Arguments

...	setSharedObjectOptions: the options you want to set, it can be copyOnWrite, sharedSubset and sharedCopy. getSharedObjectOptions: A character vector. If empty, all options will be returned.

Value

setSharedObjectOptions: No return value getSharedObjectOptions: A list of the package options or a single value

Examples

```
getSharedObjectOptions()
setSharedObjectOptions(copyOnWrite = FALSE)
getSharedObjectOptions()
getSharedObjectOptions("copyOnWrite")
```

---

share *Create a shared object*

Description

This function will create a shared object in the shared memory for the object x. There is no duplication of the shared object when it is exported to the other processes.

Usage

```
share(x, ...)

## S4 method for signature 'ANY'
share(x, ..., copyOnWrite, sharedSubset, sharedCopy, mustWork)
```

Arguments

x	An R object that you want to shared, see details.
...	For generalization purpose.
copyOnWrite, sharedSubset, sharedCopy	The parameters controlling the behavior of the shared object, see details.
mustWork	Whether to throw an error if x is not a sharable object(e.g. Character). This parameter has no effect on the S4 object.

Details

The function returns a shared object corresponding to the argument x if it is sharable. There should be no different between x and the return value except that the latter one is shared. The attribute(s) of x will also be shared.

Supported types

For the basic R types, the function supports raw,logical ,integer, double, complex. character cannot be shared for it has a complicated data structure and closely relates to R's cache. For the containers, the function supports list, pairlist and environment. Note that sharing a container is equivalent sharing all elements in the container, the container itself will not be shared.

The function share is an S4 generic. The default share method works for most S4 objects. Therefore, there is no need to define a S4 share method for each S4 class unless the S4 class has a special implementation(e.g. on-disk data). The default method will share the object itself and all slots it contains. No error will be given if any of these objects are not sharable and they will be kept unchanged.

Behavior control

In the R level, the behaviors of an ALTREP object is exactly the same as an atomic object but the data of an ALTREP object is allocated in the shared memory space. Therefore an ALTREP object can be easily exported to the other R processes without dulplicating the data, which reduces the memory usage and the overhead of data transmission.

The behavior of a shared object can be controlled through three parameters: copyOnWrite, sharedSubset and sharedCopy.

copyOnWrite determines Whether a new R object need to be allocated when the shared object is changed. The default value is TRUE, but can be altered by passing an argument copyOnWrite = FALSE to the function.

Please note that the no-copy-on-write feature is not fully supported by R. When copyOnWrite is FALSE, a shared object might not behaves as user expects. Please refer to the example code to see the exceptions.

sharedSubset determines whether the subset of a shared object is still a shared object. The default value is FALSE, and can be changed by passing sharedSubset = TRUE to the function

At the time this documentation is being written, The shared subset feature will cause an unnecessary memory duplication in R studio. Therefore, for the performance consideration, it is better to turn the feature off when using R studio.

sharedCopy determines whether the object is still a shared object after a duplication. If copyOnWrite is FALSE, this feature is off since the duplication cannot be triggered. In current version (R 3.6), an object will be duplicated four times for creating a shared object and lead to a serious performance problem. Therefore, the default value is FALSE, user can alter it by passing sharedCopy = FALSE to the function.

Value

A shared object

Examples

```
## For vector
x <- runif(10)
so <- share(x)
x
so
```

```
## For matrix
x <- matrix(runif(10), 2, 5)
so <- share(x)
x
so

## For data frame
x <- as.data.frame(matrix(runif(10), 2, 5))
so <- share(x)
x
so

## export the object
library(parallel)
cl <- makeCluster(1)
clusterExport(cl, "so")
## check the exported object in the other process
clusterEvalQ(cl, so)

## close the connection
stopCluster(cl)

## Copy-on-write
## This is the default setting
x <- runif(10)
so1 <- share(x, copyOnWrite = TRUE)
so2 <- so1
so2[1] <- 10
## so1 is unchanged since copy-on-write feature is on.
so1
so2

## No-copy-on-write
so1 <- share(x, copyOnWrite = FALSE)
so2 <- so1
so2[1] <- 10
#so1 is changed
so1
so2

## Flaw of no-copy-on-write
## The following code changes the value of so1, highly unexpected! Please use with caution!
-so1
so1
## The reason is that the minus function trys to dulplicate so1 object,
## but the dulplicate function will return so1 itself, so the value in so1 also get changed.
```

---

unshare *Unshare a shared object*

Description

Unshare a shared object. There will be no effect if the object is not shared.

Usage

```
unshare(x)
```

Arguments

x	a shared object, or an object that contains a shared object.

Value

An unshared object

Examples

```
x1 <- share(1:10)
x2 <- unshare(x1)
is.shared(x1)
is.shared(x2)
```

Index