Package 'alabaster.matrix'

June 4, 2023

Title Load and Save Artifacts from File

Version 1.1.2

Date 2023-05-10

License MIT + file LICENSE

Description

Save matrices, arrays and similar objects into file artifacts, and load them back into memory.
This is a more portable alternative to serialization of such objects into RDS files.
Each artifact is associated with metadata for further interpretation;
downstream applications can enrich this metadata with context-specific properties.

Imports methods, BiocGenerics, DelayedArray (>= 0.27.2), rhdf5,
HDF5Array, Matrix, alabaster.base

Suggests testthat, knitr, BiocStyle, S4Vectors, chihaya

VignetteBuilder knitr

RoxygenNote 7.2.3

biocViews DataImport, DataRepresentation

git_url https://git.bioconductor.org/packages/alabaster.matrix

git_branch devel

git_last_commit ef62482

git_last_commit_date 2023-05-10

Date/Publication 2023-06-04

Author Aaron Lun [aut, cre]

Maintainer Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

R topics documented:

createRawArraySeed *Array loading utilities*

Description

Utilities for loading an array saved by `stageObject`.

Usage

```
.createRawArraySeed(info, project, names = TRUE)

.extractArrayDimnames(path, group, ndim)
```

Arguments

`info`	A named list of metadata for this array.
`project`	Any argument accepted by the acquisition functions, see ?`acquireFile`. By default, this should be a string containing the path to a staging directory.
`names`	Logical scalar indicating whether the seed should be annotated with dimnames (if available).
`path`	String containing the path to the file containing said array.
`group`	String containing the name of the group with the dimnames.
`ndim`	Integer scalar specifying the number of dimensions.

Details

For `.createArraySeed`, the array should be one of `hdf5_dense_array`, `hdf5_sparse_matrix` or `hdf5_delayed_array`.

For delayed arrays, the file may contain a seed array with the `"custom alabaster local array"` type. This should have a `path` dataset containing a relative path to another array in the same `project`, which is loaded and used as the seed for this delayed array. Callers can overwrite this behavior by setting `"custom alabaster local array"` in the `knownArrays` from **chihaya** before calling `.createRawArraySeed`.

For `.extractArrayDimnames`, `path` is expected to be a HDF5 file with a group specified by `group`. Each child of this group is a string dataset named after a (0-indexed) dimension, containing the names for that dimension.

Value

`.createRawArraySeed` returns a seed that can be used in the `DelayedArray` constructor.

`.extractArrayDimnames` returns a list of character vectors or `NULL`, containing the dimnames.

Author(s)

Aaron Lun

Examples

```
# Staging an array as an example:
dir <- tempfile()
dir.create(dir)
mat <- array(rpois(10000, 10), c(50, 20, 10))
meta <- stageObject(mat, dir, "whee")

# Loading it back as a DelayedArray seed:
.createRawArraySeed(meta, project=dir)
```

`loadArray` *Load high-dimensional arrays*

Description

Default loading of arrays from on-disk formats, using the corresponding `stageObject` method. It should not be necessary for users to call this function manually.

Usage

```
loadArray(info, project)
```

Arguments

`info`	Named list containing the metadata for this array.
`project`	Any argument accepted by the acquisition functions, see `?acquireFile`. By default, this should be a string containing the path to a staging directory.

Value

A multi-dimensional object (usually a DelayedMatrix) containing the array data.

Author(s)

Aaron Lun

Examples

```
dir <- tempfile()
dir.create(dir)

arr <- array(rpois(10000, 10), c(50, 20, 10))
dimnames(arr) <- list(
   paste0("GENE_", seq_len(nrow(arr))),
   letters[1:20],
   NULL
)
```

```
path <- "whee"
info <- stageObject(arr, dir, path)
loadArray(info, project=dir)
```

`preserveDelayedOperations` *Preserve delayed operations during staging*

Description

Preserve delayed operations via **chihaya** when staging a DelayedArray with `stageObject`.

Usage

```
preserveDelayedOperations(preserve)
```

Arguments

`preserve`	Whether to preserve delayed operations using the **chihaya** specification.

Details

By default, any DelayedArray in `stageObject` will be saved as a new dense array or sparse matrix. However, if this option is enabled, DelayedArrays will instead be saved in the **chihaya** specification, where the delayed operations are themselves stored in the HDF5 file (see https://ltla.github.io/chihaya for details).

The **chihaya** specification is more complicated to parse but can be helpful in reducing disk usage. One simple example is to avoid sparsity-breaking or integer-to-float operations by storing their delayed representations in the file. If the seed matrix is derived from some immutable reference location, advanced users can even store links to that location instead of duplicating the seed data.

Value

Logical scalar indicating whether delayed operations are to be preserved by the DelayedArray method. If `preserve` is supplied, it is used to set this scalar, and the *previous* value of the scalar is invisibly returned.

Author(s)

Aaron Lun

Examples

```
preserveDelayedOperations()
old <- preserveDelayedOperations(TRUE)
preserveDelayedOperations()
preserveDelayedOperations(old)
```

`recycleHdf5Files` *Recycle existing HDF5 files*

Description

Re-use existing files in HDF5-backed arrays rather than reserializing them in `stageObject`.

Usage

```
recycleHdf5Files(recycle)
```

Arguments

`recycle`	Whether to recycle existing files for HDF5-backed DelayedArrays.

Details

If this options is enabled, `stageObject` will attempt to link/copy existing files for any HDF5-backed DelayedArray instances - most specifically, HDF5Array objects and H5SparseMatrix objects using the 10X format. This avoids re-serialization of the data for faster staging. It also allows advanced users to add their own customizations into the HDF5 file during staging, as long as they do not interfere with `loadArray`.

By default, this option is disabled as the properties of the existing file are not known in the general case. In particular, the file might contain other groups/datasets that are irrelevant, and use up extra disk space if copied; or confidential, and should not be stored in the staging directory. Users should only enable this option if they have full control over the generation and contents of the backing HDF5 files.

Also note that any dimnames on `x` will be ignored during recycling.

Value

Logical scalar indicating whether HDF5 files are to be reused. If `recycle` is supplied, it is used to set this scalar, and the *previous* value of the scalar is invisibly returned.

Author(s)

Aaron Lun

Examples

```
recycleHdf5Files()
old <- recycleHdf5Files(TRUE)
recycleHdf5Files()
recycleHdf5Files(old)
```

stageArray *Stage a multi-dimensional array for upload*

Description

Stage a high-dimensional array in preparation for upload to DataSetDB.

Usage

```
## S4 method for signature 'array'
stageObject(x, dir, path, child = FALSE)

## S4 method for signature 'DelayedArray'
stageObject(x, dir, path, child = FALSE)

## S4 method for signature 'Matrix'
stageObject(x, dir, path, child = FALSE)

## S4 method for signature 'DelayedMatrix'
stageObject(x, dir, path, child = FALSE)
```

Arguments

x	An array, almost always integer or numeric, though logical and character matrices are also supported. Alternatively, a DelayedArray or any instance of a Matrix class.
dir	String containing the path to the staging directory.
path	String containing the relative path to a subdirectory inside the staging directory, in which x is to be saved.
child	Logical scalar indicating whether x is a child of a larger object.

Details

For dense arrays, we save the array as a dense matrix in a HDF5 file using methods from the **HDF5Array** package. For sparse matrices, we call `writeSparseMatrix` to save the data in the 10X sparse matrix format. Other representations may have more appropriate formats, which are supported by simply writing new methods for this generic. Note that specialized methods will usually require new schemas to validate any new metadata fields.

If x itself is a child of a larger object, we suggest using the output path when referencing x from within the larger object's metadata. This is because stageObject methods may add more path components, file extensions, etc. to the input path when saving the object. As a result, the output path may not be the same as the input path.

Value

x is saved into a single file at file.path(dir, path), possibly after appending an arbitrary file extension. A named list is returned, containing at least:

- $schema, a string specifying the schema to use to validate the metadata.
- path, a string containing the path to the file inside the subdirectory, containing the assay contents.
- is_child, a logical scalar equal to the input child.

Author(s)

Aaron Lun

See Also

preserveDelayedOperations, to preserve the delayed'ness of a DelayedMatrix x.

recycleHdf5Files, to re-use the existing file in a HDF5-backed DelayedMatrix x.

Examples

```
dir <- tempfile()
dir.create(dir)

mat <- array(rpois(10000, 10), c(50, 20, 10))
dimnames(mat) <- list(
   paste0("GENE_", seq_len(nrow(mat))),
   letters[1:20],
   NULL
)

path <- "whee"
stageObject(mat, dir, path)

list.files(dir)
```

WrapperArraySeed *DelayedArray wrapper seed*

Description

Virtual class for a DelayedArray wrapper seed. This automatically forwards DelayedArray generic operations onto an internal seed class. Concrete subclasses are expected to attach more provenance-tracking information, while the internal seed handles the heavy lifting of data extraction, e.g., H5SparseMatrixSeed or HDF5ArraySeed objects.

Subclass developers can also create methods for the loadWrapperArray generic. This should accept two arguments:

- `meta`, a list containing metadata for the array.
- `project`, an object specifying the project of interest. This is the sole argument used for S4 dispatch.

It should then return an instance of a WrapperArray subclass that retains some provenance about the resource from which it was generated.

Examples

```
# Mocking up a concrete wrapper array class, which contains an
# extra 'foo_id' slot to track the provenance of the data.
setClass("FooArraySeed", contains="WrapperArraySeed",
    slots=c(seed="ANY", foo_id="character"))

y <- Matrix::rsparsematrix(1000, 100, 0.01)
foo <- new("FooArraySeed", seed=y, foo_id="FOO.0001")

dim(foo)
is_sparse(foo)
extract_array(foo, list(1:10, 1:10))
OLD_extract_sparse_array(foo, list(1:10, 1:10))
```

`writeSparseMatrix` *Write a sparse matrix*

Description

Writes a sparse matrix to file in a compressed sparse format.

Usage

```
writeSparseMatrix(
  x,
  file,
  name,
  chunk = 10000,
  column = TRUE,
  tenx = FALSE,
  guess.integer = TRUE
)
```

Arguments

`x`	A sparse matrix of some sort. This includes sparse DelayedMatrix objects.
`file`	String containing a path to the HDF5 file. The file is created if it is not already present.
`name`	String containing the name of the group to store `x`.

chunk	Integer scalar specifying the chunk size for the indices and values.
column	Logical scalar indicating whether to store as compressed sparse column format.
tenx	Logical scalar indicating whether to use the 10X compressed sparse column format.
guess.integer	Logical scalar specifying whether to guess an appropriate integer type from x.

Details

This writes a sparse matrix to file in various formats:

- column=TRUE and tenx=FALSE uses H5AD's csr_matrix format.
- column=FALSE and tenx=FALSE uses H5AD's csc_matrix format.
- tenx=TRUE uses 10X Genomics' HDF5 matrix format.

For the first two formats, the apparent transposition is deliberate, because columns in R are interpreted as rows in H5AD. This allows us to retain consistency the interpretation of samples (columns in R, rows in H5AD) and features (vice versa). Constructors for classes like H5SparseMatrix will automatically transpose so no extra work is required.

If guess.integer=TRUE, we attempt to save x's values into the smallest type that will accommodate all of its values. If x only contains unsigned integers, we will attempt to save either 8-, 16- or 32-bit unsigned integers. If x contains signed integers, we will fall back to 32-bit signed integers. For all other values, we will fall back to double-precision floating point values.

We attempt to save x's indices to unsigned 16-bit integers if the relevant dimension of x is small enough. Otherwise we will save it as an unsigned 32-bit integer.

Value

A NULL invisibly. The contents of x are written to name in file.

Author(s)

Aaron Lun

Examples

```
library(Matrix)
x <- rsparsematrix(100, 20, 0.5)
tmp <- tempfile(fileext=".h5")
writeSparseMatrix(x, tmp, "csc_matrix")
writeSparseMatrix(x, tmp, "csr_matrix", column=FALSE)
writeSparseMatrix(x, tmp, "tenx_matrix", tenx = TRUE)

rhdf5::h5ls(tmp)
library(HDF5Array)
H5SparseMatrix(tmp, "csc_matrix")
H5SparseMatrix(tmp, "csr_matrix")
H5SparseMatrix(tmp, "tenx_matrix")
```

Index