Package ‘dipsaus’

May 12, 2020

Type Package

Title A Dipping Sauce for Data Analysis and Visualizations

Version 0.0.7

Description Works as an "add-on" to packages like 'shiny', 'future', as well as 'rlang', and provides utility functions. Just like dipping sauce adding flavors to potato chips or pita bread, 'dipsaus' for data analysis and visualizations adds handy functions and enhancements to popular packages. The goal is to provide simple solutions that are frequently asked for online, such as how to synchronize 'shiny' inputs without freezing the app, or how to get memory size on 'Linux' or 'MacOS' system. The enhancements roughly fall into these four categories: 1. 'shiny' input widgets; 2. high-performance computing using 'RcppParallel' and 'future' package; 3. modify R calls and convert among numbers, strings, and other objects. 4. utility functions to get system information such like CPU chip-set, memory limit, etc.

URL https://github.com/dipterix/dipsaus

BugReports https://github.com/dipterix/dipsaus/issues

License GPL-3

Encoding UTF-8

LazyData true

Language en-US

Depends R (>= 3.5.0)

Imports utils, grDevices, parallel, Rcpp, RcppParallel, R6, shiny, cli, stringr, jsonlite (>= 1.6), future, future.apply, progressr, crayon, fastmap, base64url, base64enc, synchronicity, digest, rlang (>= 0.4.0), startup

Suggests microbenchmark, knitr, rmarkdown, testthat, promises, magrittr, glue, qs, RcppRedis, future.callr

RoxygenNote 7.1.0

LinkingTo Rcpp, RcppParallel

VignetteBuilder knitr
NeedsCompilation: yes

Author: Zhengjia Wang [aut, cre]

Maintainer: Zhengjia Wang <zhengjia.wang@rice.edu>

Repository: CRAN

Date/Publication: 2020-05-12 09:30:06 UTC

R topics documented:

AbstractMap ......................................................... 3
AbstractQueue ....................................................... 3
actionButtonStyled .................................................. 6
ask_or_default ....................................................... 7
ask_yesno ............................................................. 8
async ................................................................. 9
async_expr ............................................................ 9
async_flapply ........................................................ 10
base64_to_image .................................................... 11
baseline_array ...................................................... 11
cat2 ................................................................. 14
check_installed_packages .......................................... 15
col2hexStr ........................................................... 15
collapse ............................................................. 16
compoundInput2 .................................................... 17
decorate_function .................................................. 20
deparse_svec ....................................................... 21
dev_create .......................................................... 22
do_aggregate ......................................................... 23
drop_nulls ........................................................... 24
eval_dirty ............................................................. 25
fastmap2 ............................................................ 26
forelse ................................................................. 28
getInputBinding ..................................................... 29
get_cpu ............................................................... 30
get_dots .............................................................. 31
get_ram ............................................................... 32
handler_dipsaus_progress ......................................... 32
iapply ................................................................. 34
lapply_async2 ...................................................... 35
list_to_fastmap2 ................................................... 36
lock ................................................................. 37
make_async_evaluator ............................................. 38
make_forked_clusters ............................................. 39
map ................................................................. 40
mask_function2 .................................................... 43
MasterEvaluator ..................................................... 44
match_calls ........................................................ 44
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem_limit2</td>
<td>45</td>
</tr>
<tr>
<td>new_function2</td>
<td>45</td>
</tr>
<tr>
<td>no_op</td>
<td>47</td>
</tr>
<tr>
<td>package_installed</td>
<td>48</td>
</tr>
<tr>
<td>parse_svec</td>
<td>49</td>
</tr>
<tr>
<td>PersistContainer</td>
<td>50</td>
</tr>
<tr>
<td>prepare_install</td>
<td>51</td>
</tr>
<tr>
<td>progress2</td>
<td>52</td>
</tr>
<tr>
<td>registerInputBinding</td>
<td>54</td>
</tr>
<tr>
<td>screenshot</td>
<td>55</td>
</tr>
<tr>
<td>set_shiny_input</td>
<td>56</td>
</tr>
<tr>
<td>shift_array</td>
<td>57</td>
</tr>
<tr>
<td>sync_shiny_inputs</td>
<td>58</td>
</tr>
<tr>
<td>time_delta</td>
<td>59</td>
</tr>
<tr>
<td>to_datauri</td>
<td>60</td>
</tr>
<tr>
<td>to_ram_size</td>
<td>61</td>
</tr>
<tr>
<td>updateActionButtonStyled</td>
<td>61</td>
</tr>
<tr>
<td>updateCompoundInput2</td>
<td>62</td>
</tr>
<tr>
<td>update_fastmap2</td>
<td>63</td>
</tr>
<tr>
<td>use_shiny_dipsaus</td>
<td>64</td>
</tr>
<tr>
<td>%=&gt;%</td>
<td>64</td>
</tr>
<tr>
<td>%?&lt;-%</td>
<td>65</td>
</tr>
<tr>
<td>%+-%</td>
<td>66</td>
</tr>
</tbody>
</table>

**AbstractMap**

*Abstract Map to store key-value pairs*

**Description**

Abstract Map to store key-value pairs

**AbstractQueue**

*Defines abstract queue class*

**Description**

This class is inspired by [https://cran.r-project.org/package=txtq](https://cran.r-project.org/package=txtq). The difference is AbstractQueue introduce an abstract class that can be extended and can queue not only text messages, but also arbitrary R objects, including expressions and environments. All the queue types in this package inherit this class.
Abstract Public Methods

Methods start with @... are not thread-safe. Most of them are not used directly by users. However, you might want to override them if you inherit this abstract class. Methods marked as "(override)" are not implemented, meaning you are supposed to implement the details. Methods marked as "(optional)" usually have default alternatives.

initialize(...) (override) The constructor. Usually three things to do during the process: 1. set get_locker free_locker if you don't want to use the default lockers. 2. set lock file (if using default lockers). 3. call self$connect(...)

get_locker(), free_locker() (optional) Default is NULL for each methods, and queue uses an internal private$default_get_locker and private$default_free_locker. These two methods are for customized locker, please implement these two methods as functions during self$initialization get_locker obtains and lock access (exclusive), and free_locker frees the locker. Once implemented, private$exclusive will take care the rest. Type: function; parameters: none; return: none

@get_head(), @set_head(v) (override) Get head so that we know where we are in the queue self$@get_head() should return a integer indicating where we are at the queue self$@set_head(v) stores that integer. Parameter v is always non-negative, this is guaranteed. Users are not supposed to call these methods directly, use self$head and self$head<- instead. However, if you inherit this class, you are supposed to override the methods.

@get_total(), @set_total(v) (override) Similar to @get_head and @set_head, defines the total items ever stored in the queue. total-head equals current items in the queue.

@inc_total(n=1) (optional) Increase total, usually this doesn't need to be override, unless you are using files to store total and want to decrease number of file connections

@append_header(msg, ...) (override) msg will be vector of strings, separated by "|", containing encoded headers: 'time', 'key', 'hash', and 'message'. to decode what's inside, you can use self$print_items(stringr::str_split_fixed(msg, '"Var"|"Var",4)). Make sure to return a number, indicating number of items stored. Unless handled elsewhere, usually return(length(msg)).

@store_value(value, key) (override) Defines how to store value. 'key' is unique identifier generated from time, queue ID, and value. Usually I use it as file name or key ID in database. value is an arbitrary R object to store. you need to store value somewhere and return a string that will be passed as 'hash' in self$restore_value.

restore_value(hash, key, preserve = FALSE) (override) Method to restore value from given combination of 'hash' and 'key'. 'hash' is the string returned by @store_value, and 'key' is the same as key in @store_value. preserve is a indicator of whether to preserve the value for future use. If set to FALSE, then you are supposed to free up the resource related to the value. (such as free memory or disk space)

@log(n = -1, all = FALSE) (override) get n items from what you saved to during @append_header. n less equal than 0 means listing all possible items. If all=TRUE, return all items (number of rows should equals to self$total), including popped items. If all=FALSE, only return items in the queue (number of rows is self$count). The returned value should be a n x 4 matrix. Usually I use stringr::str_split_fixed(...,'\|',4). Please see all other types implemented for example.

@reset(...) (override) Reset queue, remove all items and reset head, total to be 0.

@clean() (override) Clean the queue, remove all the popped items.
AbstractQueue

@validate() (override) Validate the queue. Stop if the queue is broken.

@connect(con, ...) (override) Set up connection. Usually should be called at the end of self$initialize to connect to a database, a folder, or an existing queue you should do checks whether the connection is new or it's an existing queue.

connect(con, ...) (optional) Thread-safe version. Sometimes you need to override this function instead of @connect, because private$exclusive requires lockfile to exist and to be locked. If you don't have lockers ready, or need to set lockers during the connection, override this one.

destroy() (optional) Destroy a queue, free up space and call delayedAssign('.lockfile',{stop(...)},assign.env=private) to raise error if a destroyed queue is called again later.

Public Methods

Usually don't need to override unless you know what you are doing.

push(value, message='',...) Function to push an arbitrary R object to queue. message is a string giving notes to the pushed item. Usually message is stored with header, separated from values. The goal is to describe the value. ... is passed to @append_header

pop(n = 1, preserve = FALSE) Pop n items from the queue. preserve indicates whether not to free up the resources, though not always guaranteed.

print_item(item), print_items(items) To decode matrix returned by log(), returning named list or data frame with four heads: 'time', 'key', 'hash', and 'message'.

list(n=-1) List items in the queue, decoded. If n is less equal than 0, then list all results. The result is equivalent to self$print_items(self$log(n))

log(n=-1,all=FALSE) List items in the queue, encoded. This is used with self$print_items. When all=TRUE, result will list the records ever pushed to the queue since the last time queue is cleaned. When all=FALSE, results will be items in the queue. n is the number of items.

Public Active Bindings

id Read-only property. Returns unique ID of current queue.

lockfile The lock file.

head Integer, total number of items popped, i.e. inactive items.

total Total number of items ever pushed to the queue since last cleaned, integer.

count Integer, read-only, equals to total - head, number of active items in the queue

Private Methods or properties

.id Don't use directly. Used to store queue ID.

.lockfile Location of lock file.

lock Preserve the file lock.

exclusive(expr,...) Function to make sure the methods are thread-safe

default_get_locker() Default method to lock a queue

default_free_locker Default method to free a queue
**Description**

Action Button but with customized styles

**Usage**

```r
actionButtonStyled(
    inputId,
    label,
    icon = NULL,
    width = NULL,
    type = "primary",
    btn_type = "button",
    class = "",
    ...
)
```

**Arguments**

- `inputId`, `label`, `icon`, `width`, ...
  - passed to `shiny::actionButton`
- `type`
  - button type, choices are 'default', 'primary', 'info', 'success', 'warning', and 'danger'
- `btn_type`
  - HTML tag type, either "button" or "a"
- `class`
  - additional classes to be added to the button

**Value**

'HTML' tags

**See Also**

`updateActionButtonStyled` for how to update the button.

**Examples**

```r
# demo('example-actionButtonStyled', package='dipsaus')

library(shiny)
library(dipsaus)

ui <- fluidPage(
  actionButtonStyled('btn', label = 'Click me', type = 'default'),
)
ask_or_default

- Read a Line from the Terminal, but with Default Values

Description

Ask a question and read from the terminal in interactive scenario

Usage

```r
ask_or_default(..., default = "", end = "", level = "INFO")
```

Arguments

...\nend, level passed to `cat2`

default default value to return in case of blank input

Details

The prompt string will ask a question, providing defaults. Users need to enter the answer. If the answer is blank (no space), then returns the default, otherwise returns the user input.

This can only be used in an interactive session.

Value

A character from the user’s input, or the default value. See details.
See Also

cat2, readline, ask_yesno

Examples

```r
if(interactive()){
  ask_or_default('What is the best programming language?',
                 default = 'PHP')
}
```

```r
ask_yesno

Ask and Return True or False from the Terminal

Description

Ask a question and read from the terminal in interactive scenario

Usage

```
ask_yesno(..., end = "", level = "INFO", error_if_canceled = TRUE)
```

Arguments

```
..., end, level passed to cat2
error_if_canceled
  raise error if canceled.
```

Details

The prompt string will ask for an yes or no question. Users need to enter "y", "yes" for yes, "n", "no" or no, and "c" for cancel (case-insensitive).

This can only be used in an interactive session.

Value

logical or NULL or raise an error. If "yes" is entered, returns TRUE; if "no" is entered, returns FALSE; if "c" is entered, error_if_canceled=TRUE will result in an error, otherwise return NULL

See Also

cat2, readline, ask_or_default

Examples

```
if(interactive()){
  ask_yesno('Do you know how hard it is to submit an R package and ',
            'pass the CRAN checks?')
  ask_yesno('Can I pass the CRAN check this time?')
}
```
async

Evaluate expression in async_expr

Description
Evaluate expression in async_expr

Usage
async(expr)

Arguments
expr R expression

See Also
async_expr

async_expr
Apply R expressions in a parallel way

Description
Apply R expressions in a parallel way

Usage
async_expr(
  .X,
  .expr,
  .varname = "x",
  envir = parent.frame(),
  .pre_run = NULL,
  .ncore = future::availableCores(),
  ...
)

Arguments
.X a vector or a list to apply evaluation on
.expr R expression, unquoted
.varname variable name representing element of each .X
.envir environment to evaluate expressions
.pre_run expressions to be evaluated before looping.
.ncore number of CPU cores
... passed to future::future
async_flapply

Details

async_expr uses lapply and future::future internally. Within each loop, an item in ".X" will be assigned to variable "x" (defined by ".varname") and enter the evaluation. During the evaluation, function async is provided. Expressions within async will be evaluated in another session, otherwise will be evaluated in current session. Below is the workflow:

- Run .pre_run
- For i in seq_along(.X):
  - 1. Assign x with .X[[i]], variable name x is defined by .varname
  - 2. Evaluate expr in current session.
    * a. If async is not called, return evaluated expr
    * b. If async(async_expr) is called, evaluate async_expr in another session, and return the evaluation results if async_expr

Value

a list whose length equals to .X. The value of each item returned depends on whether async is called. See details for workflow.

async_flapply       Wrapper for future.apply::future_lapply

Description

Wrapper for future.apply::future_lapply

Usage

async_flapply(X, FUN, ...)

Arguments

X, FUN, ...    passing to future.apply::future_lapply

See Also

future_lapply
**base64_to_image**  
*Save "Base64" Data to Images*

**Description**

Save "Base64" Data to Images

**Usage**

```r
base64_to_image(data, path)
```

**Arguments**

- `data`: characters, encoded "Base64" data for images
- `path`: file path to save to

**Value**

Absolute path of the saved file

---

**baseline_array**  
*Calculate Contrasts of Arrays in Different Methods*

**Description**

Provides five methods to baseline an array and calculate contrast.

**Usage**

```r
baseline_array(
  x,
  along_dim,
  baseline_indexpoints,
  unit_dims = seq_along(dim(x))[-along_dim],
  method = c("percentage", "sqrt_percentage", "decibel", "zscore", "sqrt_zscore")
)
```

**Arguments**

- `x`: array (tensor) to calculate contrast
- `along_dim`: integer range from 1 to the maximum dimension of `x`. baseline along this dimension, this is usually the time dimension.
- `baseline_indexpoints`: integer vector, which index points are counted into baseline window? Each index ranges from 1 to `dim(x)[[along_dim]]`. See Details.
unit_dims integer vector, baseline unit: see Details.
method character, baseline method options are: "percentage", "sqrt_percentage", "decibel", "zscore", and "sqrt_zscore"

Details
Consider a scenario where we want to baseline a bunch of signals recorded from different locations. For each location, we record \( n \) sessions. For each session, the signal is further decomposed into frequency-time domain. In this case, we have the input \( x \) in the following form:

\[
\text{session} \times \text{frequency} \times \text{time} \times \text{location}
\]

Now we want to calibrate signals for each session, frequency and location using the first 100 time points as baseline points, then the code will be

\[
\text{baseline} \_ \text{array}(x, \text{along} \_ \text{dim} = 3, 1 : 100, \text{unit} \_ \text{dims} = c(1, 2, 4))
\]

\( \text{along} \_ \text{dim} = 3 \) is dimension of time, in this case, it’s the third dimension of \( x \). \( \text{baseline} \_ \text{indexpoints} = 1 : 100 \), meaning the first 100 time points are used to calculate baseline. \( \text{unit} \_ \text{dims} \) defines the unit signal. Its value \( c(1, 2, 4) \) means the unit signal is per session (first dimension), per frequency (second) and per location (fourth).

In some other cases, we might want to calculate baseline across frequencies then the unit signal is \( \text{frequency} \times \text{time} \), i.e. signals that share the same session and location also share the same baseline. In this case, we assign \( \text{unit} \_ \text{dims} = c(1, 4) \).

There are five baseline methods. They fit for different types of data. Denote \( z \) is an unit signal, \( z_0 \) is its baseline slice. Then these baseline methods are:

"percentage"

\[
\frac{z - z_0}{z_0} \times 100\%
\]

"sqrt_percentage"

\[
\frac{\sqrt{z} - \sqrt{z_0}}{\sqrt{z_0}} \times 100\%
\]

"decibel"

\[
10 \times (\log_{10}(z) - \log_{10}(z_0))
\]

"zscore"

\[
\frac{z - \bar{z}_0}{sd(z_0)}
\]

"sqrt_zscore"

\[
\frac{\sqrt{z} - \sqrt{z_0}}{sd(\sqrt{z_0})}
\]

Value
Contrast array with the same dimension as \( x \).
Examples

library(dipsaus)
set.seed(1)

# Generate sample data
dims = c(10,20,30,2)
x = array(rnorm(prod(dims))^2, dims)

# Set baseline window to be arbitrary 10 timepoints
baseline_window = sample(30, 10)

# ----- baseline percentage change ------

# Using base functions
re1 <- aperm(apply(x, c(1,2,4), function(y){
m <- mean(y[baseline_window])
(y/m - 1) * 100
}), c(2,3,1,4))

# Using dipsaus
re2 <- baseline_array(x, 3, baseline_window, c(1,2,4),
method = 'sqrt_percentage')

# Check different, should be very tiny (double precisions)
range(re2 - re1)

# Check speed for large dataset
if(interactive()){
dims = c(200,20,300,2)
x = array(rnorm(prod(dims))^2, dims)
# Set baseline window to be arbitrary 10 timepoints
baseline_window = seq_len(100)
f1 <- function(){
aperm(apply(x, c(1,2,4), function(y){
m <- mean(y[baseline_window])
(y/m - 1) * 100
}), c(2,3,1,4))
}
f2 <- function(){
# equivalent as bl = x[,,baseline_window,]
#
baseline_array(x, along_dim = 3,
# baseline_indexpoints = baseline_window,
# unit_dims = c(1,2,4), method = 'sqrt_percentage')
}
microbenchmark::microbenchmark(f1(), f2(), times = 3L)
}
cat2  

Color Output

Description

Color Output

Usage

cat2(
  ..., level = "DEBUG",
  print_level = FALSE,
  file = "",
  sep = " ",
  fill = FALSE,
  labels = NULL,
  append = FALSE,
  end = "\n",
  pal = list(DEBUG = "grey60", INFO = "#1d9f34", WARNING = "#ec942c", ERROR = "#f02c2c", FATAL = "#763053", DEFAULT = "grey60"),
  use_cli = TRUE,
  bullet = "auto"
)

Arguments

... 
level 'DEBUG', 'INFO', 'WARNING', 'ERROR', or 'FATAL' (total 5 levels)
print_level if true, prepend levels before messages
file, sep, fill, labels, append pass to base::cat
end character to append to the string
pal a named list defining colors see details
use_cli logical, whether to use package 'cli'
bullet character, if use 'cli', which symbol to show. see symbol

Details

There are five levels of colors by default: 'DEBUG', 'INFO', 'WARNING', 'ERROR', or FATAL. Default colors are: 'DEBUG' (grey60), 'INFO' (#1d9f34), 'WARNING' (#ec942c), 'ERROR' (#f02c2c), 'FATAL' (#763053) and 'DEFAULT' (#000000, black). If level is not in preset five levels, the color will be "default"-black color.

Value

none.
check_installed_packages

Check If Packages Are Installed, Returns Missing Packages

Usage

check_installed_packages(
  pkgs,
  libs = base::libPaths(),
  auto_install = FALSE,
  ...
)

Arguments

- pkgs: vector of packages to install
- libs: paths of libraries
- auto_install: automatically install packages if missing
- ...: other parameters for `install.packages`

Value

package names that are not installed

col2hexStr

Convert color to Hex string

Description

Convert color to Hex string

Usage

col2hexStr(col, alpha = NULL, prefix = "#", ...)

Arguments

- col: character or integer indicating color
- alpha: NULL or numeric, transparency. See `grDevices::rgb`
- prefix: character, default is "#"
- ...: passing to `adjustcolor`
Details

col2hexStr converts colors such as 1, 2, 3, "red", "blue", ... into hex strings that can be easily recognized by ‘HTML’, ‘CSS’ and ‘JavaScript’. Internally this function uses adjustcolor with two differences:

1. the returned hex string does not contain alpha value if alpha is NULL;
2. the leading prefix "#" can be customized

Value

characters containing the hex value of each color. See details

See Also

adjustcolor

Examples

col2hexStr(1, prefix = '0x')  # "0x000000"
col2hexStr('blue')           # "#0000FF"

# Change default palette, see "grDevices::colors()"
grDevices::palette(c('orange3', 'skyblue1'))
col2hexStr(1)                # Instead of #000000, #CD8500

collapse                  Collapse Sensors And Calculate Summations/Mean (stable)

Description

Collapse Sensors And Calculate Summations/Mean

(stable)

Usage

collapse(x, keep, average = FALSE)

Arguments

x A numeric multi-mode tensor (array), without NA
keep Which dimension to keep
average collapse to sum or mean

Value

a collapsed array with values to be mean or summation along collapsing dimensions
Examples

# Example 1
x = matrix(1:16, 4)

# Keep the first dimension and calculate sums along the rest
collapse(x, keep = 1)
rowSums(x)  # Should yield the same result

# Example 2
x = array(1:120, dim = c(2,3,4,5))
result = collapse(x, keep = c(3,2))
compare = apply(x, c(3,2), sum)
sum(abs(result - compare))  # The same, yield 0 or very small number (1e-10)

# Example 3 (performance)
RcppParallel::setThreadOptions(numThreads = -1)  # auto multicores
# Small data, no big difference, even slower
x = array(rnorm(240), dim = c(4,5,6,2))
microbenchmark::microbenchmark(
    result = collapse(x, keep = c(3,2)),
    compare = apply(x, c(3,2), sum),
    times = 1L, check = function(v){
        max(abs(range(do.call("-", v)))) < 1e-10
    }
)

# large data big difference
x = array(rnorm(prod(300,200,105)), c(300,200,105,1))
microbenchmark::microbenchmark(
    result = collapse(x, keep = c(3,2)),
    compare = apply(x, c(3,2), sum),
    times = 1L, check = function(v){
        max(abs(range(do.call("-", v)))) < 1e-10
    }
)

compoundInput2

Compound input that combines and extends shiny inputs

Description

Compound input that combines and extends shiny inputs

Usage

compoundInput2(
    inputId,
    label = "Group",
    components = shiny::tagList(),
)
compoundInput2

```
  initial_ncomp = 1,
  min_ncomp = 0,
  max_ncomp = 10,
  value = NULL,
  label_color = 1,
  max_height = NULL,
...)
```

**Arguments**

- **inputId** character, shiny input ID
- **label** character, will show on each groups
- **components** ‘HTML’ tags that defines and combines HTML components within groups
- **initial_ncomp** numeric initial number of groups to show, non-negative
- **min_ncomp** minimum number of groups, default is 0, non-negative
- **max_ncomp** maximum number of groups, default is 10, greater or equal than min_ncomp
- **value** list of lists, initial values of each inputs, see examples.
- **label_color** integer or characters, length of 1 or max_ncomp, assigning colors to each group labels,
- **max_height** maximum height of the widget
- ... will be ignored

**Value**

‘HTML’ tags

**See Also**

updateCompoundInput2 for how to update inputs

**Examples**

```
library(shiny); library(dipsaus)
compoundInput2(
  'input_id', 'Group',
  div(
    textInput('text', 'Text Label'),
    sliderInput('sli', 'Slider Selector', value = 0, min = 1, max = 1)
  ),
  label_color = 1:10,
  value = list(
    list(text = '1'),  # Set text first group to be "1"
    list(),           # no settings for second group
    list(sli = 0.2),  # sli = 0.2 for the third group
  )
)
```

# Source - system.file("demo/example-compountInput2.R", package='dipsaus')
# demo('example-compoundInput2', package='dipsaus')

library(shiny)
library(dipsaus)
ui <- fluidPage(
  fluidRow(
    column(
      width = 4,
      compoundInput2('compound', 'Group Label', label_color = 1:10,
        components = div(
          textInput('txt', 'Text'),
          selectInput('sel', 'Select', choices = 1:10, multiple = TRUE),
          sliderInput('sli', 'Slider', max=1, min=0, val=0.5)
        ),
        value = list(
          list(txt = '1'),  # Set text first group to be "1"
          # no settings for second group
          list(sli = 0.2)  # sli = 0.2 for the third group
        ),
      hr(),
      actionButton('action', 'Update compound input')
    )
  )
)

server <- function(input, output, session) {
  observe({
    print(input$compound)
  })
  observe({
    # Getting specific input at group 1
    print(input$compound_txt_1)
  })
  observeEvent(input$action, {
    updateCompoundInput2(session, 'compound',
      # Update values for each components
      value = lapply(1:5, function(ii){
        list(
          txt = sample(LETTERS, 1),
          sel = sample(1:10, 3),
          sli = runif(1)
        )
      }), ncomp = NULL, txt = list(label = as.character(Sys.time())))
  })
}

if( interactive() ){
  shinyApp(ui, server, options = list(launch.browser = TRUE))
}
**Description**

Python-style decorator

**Usage**

decorate_function(orig, decor, ...)

lhs %D% rhs

**Arguments**

- orig, lhs: any function
- decor, rhs: decorator function that takes `orig` as its first argument
- ...: passed to `decor`

**Examples**

```r
# Example 1: basic usage
# Decorator that prints summary of results and return results itself
verbose_summary <- function(...){
  summary_args <- list(...)
  function(f){
    function(...){
      results <- f(...)

      print(do.call(
        summary, 
        c(list(results), summary_args)
      ))
      results
    }
  }
}

# runs as.list, but through verbose_summary
as_list2 <- decorate_function(as.list, verbose_summary)

# run test
```
res <- as_list2(1:3)  # will verbose summary
identical(res, as.list(1:3))

# Example 2
x <- 1:20
y <- x + rnorm(20)

# decorator, add a line with slope 1 with given intercept
abline_xy <- function(b){
  function(f){
    function(...){
      f(...)
      intercept <- get_dots('intercept', 0, ...)
      abline(a = intercept, b = b)
    }
  }
}

# orig, plot whatever x vs jittered+intercept
plot_xy <- function(x, intercept = rnorm(1)){
  plot(x, jitter(x, amount = 3) + intercept)
}

# new function that decorate plot_xy with abline_xy, and
# returns the intercept
plot_xy2 <- decorate_function(plot_xy, abline_xy, b = 1)

# alternatively, you might also want to try
plot_xy2 <- plot_xy %D% abline_xy(b = 1)

plot_xy2(x = 1:20)

---

**deparse_svec**  
*Convert Integer Vectors To String (stable)*

**Description**

Convert Integer Vectors To String  
(stable)

**Usage**

```r
deparse_svec(
  nums,
  connect = "-",
  concatenate = TRUE,
  collapse = ",",
  max_lag = 1
)
```
Arguments

- **nums**: integer vector
- **connect**: character used to connect consecutive numbers
- **concatenate**: connect strings if there are multiples
- **collapse**: if concatenate, character used to connect strings
- **max_lag**: defines "consecutive", min = 1

Value

strings representing the input vector. For example, c(1,2,3) returns "1-3".

See Also

- `parse_svec`

Examples

```r
deparse_svec(c(1:10, 15:18))
```

---

dev_create

Create a group of named graphic devices

Description

Create a group of named graphic devices

Usage

```r
dev_create(..., env = parent.frame())
```

Arguments

- **...**: named expressions to launch devices
- **env**: environment to evaluate expressions

Value

A list of functions to query, control, and switch between devices
Examples

```r
## Not run: ## Unix-specific example

# Create multiple named devices
devs <- dev_create(line = X11(), points = x11())

# switch to device named "points"
devs$dev_which('points')
plot(1:10)

# switch to "line" device
devs$dev_switch('line')
plot(1:100, type='l')

# Create another group with conflict name
dev_another <- dev_create(line = X11())

# Query device name with 'line'
dev_another$dev_which('line') # 4
devs$dev_which('line') # 2, doesn't conflict with the new groups

dev.list()
# close one or more device
dev_another$dev_off('line')
dev.list()

# close all devices
devs$dev_off()
dev.list()
```

## End(Not run)

---

**do_aggregate**

*Make aggregate pipe-friendly*

**Description**

A pipe-friendly wrapper of `aggregate` when using formula as input.

**Usage**

```r
do_aggregate(x, ...)
```

**Arguments**

- `x` an R object
- `...` other parameters passed to `aggregate`
Value

Results from aggregate

See Also

aggregate

Examples

```r
library(magrittr)
data(ToothGrowth)

ToothGrowth %>%
do_aggregate(len ~ ., mean)
```

---

drop_nulls  
_Drop NULL values from list or vectors_

Description

Drop NULL values from list or vectors

Usage

```r
drop_nulls(x, .invalids = list("is.null"))
```

Arguments

- `x` : list to check
- `.invalids` : a list of functions, or function name. Default is 'is.null'.

Value

list or vector containing no invalid values

Examples

```r
x <- list(NULL,NULL,1,2)
drop_nulls(x)  # length of 2
```
**eval_dirty**

Evaluate expressions

```r
eval_dirty(expr, env = parent.frame(), data = NULL, quoted = TRUE)
```

**Arguments**

- `expr` R expression or `rlang` quo
- `env` environment to evaluate
- `data` dataframe or list
- `quoted` Is the expression quoted? By default, this is `TRUE`. This is useful when you don’t want to use an expression that is stored in a variable; see examples

**Details**

eval_dirty uses `base::eval()` function to evaluate expressions. Compare to `rlang::eval_tidy`, which won’t affect original environment, eval_dirty causes changes to the environment. Therefore if `expr` contains assignment, environment will be changed in this case.

**Value**

the executed results of `expr` evaluated with side effects.

**Examples**

```r
env = new.env(); env$a = 1
rlang::eval_tidy(quote({a <- 111}), env = env)
print(env$a)  # Will be 1. This is because eval_tidy has no side effect

eval_dirty(quote({a <- 111}), env)
print(env$a)  # 111, a is changed

# Unquoted case
eval_dirty({a <- 222}, env, quoted = FALSE)
print(env$a)
```
Description

`fastmap` provides a key-value store where the keys are strings and the values are any R objects. It differs from normal environment that `fastmap` avoids memory leak. `fastmap2` is a wrapper for `fastmap`, which provides several generic functions such that it has similar behaviors to lists or environments.

Usage

```r
fastmap2(missing_default = NULL)

## S3 method for class 'fastmap2'
x[[name]]

## S3 method for class 'fastmap2'
x$name

## S3 replacement method for class 'fastmap2'
x[[name]] <- value

## S3 replacement method for class 'fastmap2'
x$name <- value

## S3 method for class 'fastmap2'
x[i, j = NULL, ...]

## S3 replacement method for class 'fastmap2'
x[i, j = NULL, ...] <- value

## S3 method for class 'fastmap2'
names(x)

## S3 method for class 'fastmap2'
print(x, ...)

## S3 method for class 'fastmap2'
length(x)

## S3 method for class 'fastmap2'
as.list(x, ...)
```

Arguments

- `missing_default` passed to `fastmap::fastmap`
x a 'fastmap2' object
name name, or key of the value
value any R object
i, j vector of names
... passed to other methods

Value
A list of 'fastmap2' instance

Examples

```r
## --------------------------- Basic Usage --------------------------
map <- fastmap2()
map$a = 1
map$b = 2
print(map)

map[c('a', 'b')]
# Alternative way
map['a', 'b']

map[c('c', 'd')] <- 3:4
# or
map['e', 'f'] <- 5:6

# The order is not guaranteed, unless sort=TRUE
as.list(map)
as.list(map, sort=TRUE)

names(map)
length(map)

## ----------------------- NULL value handles -----------------------
map$b <- NULL
names(map) # 'b' still exists!
as.list(map) # 'b' is NULL, but still there

# to remove 'b', you have to use '@remove' method
map$@remove('b')

## ---------------- Native fastmap::fastmap methods -----------------
# whether map has 'a'
map$@has('a')

# Remove a name from list
map$@remove('a')

# remove all from list
```
Description

Provide Python-style "for-else" that works as follows: for each element, execute "for" block, if there is break while executing "for" block, then just stop and ignore the "else" statement, otherwise run "else" block.

Usage

forelse(x, FUN, ALT_FUN = NULL)

Arguments

x
iterative R objects such as list, vector, etc.

FUN
function that applies to each x

ALT_FUN
function that takes no argument or other types of R object

Value

If any FUN returns anything other than NULL, then the function returns the first none NULL object. If all x fed to FUN return NULL, then this function returns ALT_FUN (if ALT_FUN is not a function) or the result of ALT_FUN().

Examples

# 1. ALT_FUN get executed because FUN returns NULL for all items in x
forelse(
  1:10,
  function(x){
    cat('The input is ', x, end = '\n')
    if( x > 10) return(x) else return(NULL)
  },
  function(){
    cat('ALT_FUN is executed!\n')
    'wow'
  }
)

# 2. FUN returns non-NULL object
forelse(
1:10,
function(x){
    cat(The input is ', x, end = 'n')
    if( x %% 2 == 0 ) return(x) else return(NULL)
},
'wow'

# --------------------------- Performance ------------------------------
FUN <- function(x){
    Sys.sleep(0.01)
    if( x %% 2 == 0 ) return(x) else return(NULL)
}

microbenchmark::microbenchmark({
    forelse(1:10, FUN, 'wow'
  }, {
    y <- unlist(lapply(1:10, FUN))
    if(length(y)){
        y <- y[[1]]
    }else{
        y <- 'wow'
    }
  }, {
    y <- NULL
    for(x in 1:10){ y <- FUN(x) }
    if(is.null(y)){ y <- 'wow' }
  }, times = 3)

getInputBinding Obtain registered input bindings

Description
Obtain registered input bindings

Usage
getInputBinding(fname, pkg = NULL, envir = parent.frame())

Arguments
fname input function name, character or quoted expression such as 'shiny::textInput'
or numericInput.

pkg (optional), name of package

envir environment to evaluate fname if pkg is not provided
**Value**

a list containing: 1. ‘JavaScript’ input binding name; 2. ‘R’ updating function name

**Examples**

```r
library(dipsaus)

# Most recommended usage
getInputBinding('compoundInput2', pkg = 'dipsaus')

# Other usages
getInputBinding('shiny::textInput')

getInputBinding(shiny::textInput)
getInputBinding(compoundInput2, pkg = 'dipsaus')

# Bad usage, raise errors in some cases
## Not run:
## You need to library(shiny), or set envir=asNamespace('shiny'), or pkg='shiny'
getInputBinding('textInput')
getInputBinding(textInput) # also fails

## Always fails
getInputBinding('dipsaus::compoundInput2', pkg = 'dipsaus')

## End(Not run)
```

---

### get_cpu

**Get CPU Chip-set Information**

**Description**

Get CPU Chip-set Information

**Usage**

```r
get_cpu()
```

**Value**

a list of vendor ID and CPU model name
**get_dots**

*Get element from dots ’...’*

**Description**

Get specific key values from '...' without evaluating the rest of arguments.

**Usage**

```r
get_dots(.name, .default = NULL, ...)
```

**Arguments**

- `.name` character name of the argument
- `.default` R object to return if argument not found
- `...` dots that contains argument

**Details**

One could use `list(...)[[name]]` to extract any keys from the dots. However, such way reduces code readability. If some arguments have not evaluated, `list(...)` will force evaluating them. Normally it’s fine if these expressions take little time to run, but if the expression require time to run, `get_dots` avoids unnecessary evaluations.

**Examples**

```r
# ------------------------ Basic Usage ---------------------------
plot2 <- function(...){
  title = get_dots('main', 'There is no title', ...)
  plot(...)
  title
}
plot2(1:10)
plot2(1:10, main = 'Scatter Plot of 1:10')

# ------------------------ Comparisons ----------------------------
f1 <- function(...){ get_dots('x', ...) }
f2 <- function(...){ list(...)[['x']] }
delayedAssign('y', { cat('y is evaluated!') })

# y will not evaluate
f1(x = 1, y = y)

# y gets evaluated
f2(x = 1, y = y)
```
# ------------------ Decorator example -------------------

```r
ret_range <- function(which_range = 'y') {
  function(f) {
    function(...) {
      f(...)
      y_range <- range(get_dots(which_range, 0, ...))
      y_range
    }
  }
}
```

```r
plot_ret_yrange <- plot %D% ret_range('y')
plot_ret_yrange(x = 1:10, y = rnorm(10))
```

---

**get_ram**  
*Get Memory Size*

**Description**

Get Memory Size

**Usage**

```r
get_ram()
```

**Value**

Numeric in Bytes how big your system RAM is

---

**handler_dipsaus_progress**  
*Progress-bar Handler*

**Description**

Handler for *progress2* to support *progressr::handlers*. See examples for detailed use case

**Usage**

```r
handler_dipsaus_progress(
  title = getOption("dipsaus.progressr.title", "Progress"),
  intrusiveness = getOption("progressr.intrusiveness.gui", 1),
  target = if (is.null(shiny::getDefaultReactiveDomain())) "terminal" else "gui",
  ...
)
```
handler_dipsaus_progress

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>default title of progress2</td>
</tr>
<tr>
<td>intrusiveness</td>
<td>A non-negative scalar on how intrusive (disruptive) the reporter to the user</td>
</tr>
<tr>
<td>target</td>
<td>where progression updates are rendered</td>
</tr>
</tbody>
</table>

Examples

library(progressr)
library(shiny)
library(future)

## ------------------------------ Setup! ----------------------------------------
handlers(handler_dipsaus_progress())

# ------------------------------ A simple usage -----------------------------
x1 <- 1:5
handlers(handler_dipsaus_progress())
with_progress({
  p <- progressor(along = xs)
  y <- lapply(xs, function(x) {
    p(sprintf("x=%g", x))
    Sys.sleep(0.1)
    sqrt(x)
  })
})

# ------------------------ A future.apply case -----------------------------
plan(sequential)
# test it yourself with plan(multisession)

handlers(handler_dipsaus_progress())
with_progress({
  p <- progressor(along = xs)
  y <- future.apply::future_lapply(xs, function(x) {
    p(sprintf("x=%g", x))
    Sys.sleep(0.1)
    sqrt(x)
  })
})

# ------------------------ A shiny case -------------------------------------

ui <- fluidPage(
  actionButton('ok', 'Run Demo')
)

server <- function(input, output, session) {
  handlers(handler_dipsaus_progress())
make_forked_clusters()

observeEvent(input$ok, {
  with_progress(
    p <- progressor(along = 1:100)
    y <- future.apply::future_lapply(1:100, function(x) {
      p(sprintf("Input %d|Result %d", x, x+1))
      Sys.sleep(1)
      x+1
    } 
  )))
})
}
}

if(interactive()){ 
  shinyApp(ui, server)
}

---

iapply

Apply each elements with index as second input

Description

Apply function with an index variable as the second input.

Usage

iapply(X, FUN, ..., .method = c("sapply", "lapply", "vapply"))

Arguments

X a vector (atomic or list)
FUN the function to be applied to each element of X: see ‘Details’.
... passed to apply methods
.method method to use, default is sapply

Details

FUN will be further passed to the apply methods. Unlike lapply, FUN is expected to have at least two arguments. The first argument is each element of X, the second argument is the index number of the element.

Value

a list or matrix depends on .method. See lapply
**lapply_async2**  
*Apply, but in parallel*

**Description**

Apply, but in parallel

**Usage**

```
lapply_async2(
  x,
  FUN,
  FUN.args = list(),
  callback = NULL,
  plan = TRUE,
  future.chunk.size = NULL,
  ...
)
```

**Arguments**

- `x` vector, list
- `FUN` function to apply on each element of `x`
- `FUN.args` more arguments to feed into `FUN`
- `callback` function to run after each iteration
- `plan` logical, or character or future plan; see Details.
- `future.chunk.size` see also `future_eapply`. If you want the callbacks to be called immediately after each loop, then set it to 1, which is not optimal but the only way right now.
- `...` passed to `plan`

**Details**

When `plan` is logical, FALSE means use current plan. If `plan=TRUE`, then it equals to `plan='multicore'`. For characters, `plan` can be 'multicore', 'callr', 'sequential', 'multisession', 'multiprocess', etc. Alternatively, you could pass future `plan` objects.

**Value**

same as with(`FUN.args`, `lapply(x, function(el) {eval(body(FUN))})`)
Examples

```r
library(future)
plan(sequential)

# Use sequential plan
# 1. Change `plan` to 'multicore', 'multisession', or TRUE to enable
# multi-core, but still with progress information
# 2. Change plan=FALSE will use current future plan
res <- lapply_async2(100:200, function(x){
  return(x+1)
}, callback = function(e){
  sprintf('Input=%d', e)
}, plan = 'sequential')

# Disable callback message, then the function reduce to
# normal `future.apply::future_lapply`
res <- lapply_async2(100:200, function(x){
  return(x+1)
}, callback = NULL, plan = FALSE)
```

---

**list_to_fastmap2**

*Copy elements to fastmap2*

**Description**

Copy elements to fastmap2

**Usage**

```r
list_to_fastmap2(li, map = NULL)
```

**Arguments**

- **li**: a list or an environment
- **map**: NULL or a fastmap2 instance

**Value**

If map is not NULL, elements will be added to map and return map, otherwise create a new instance.
Description

A wrapper for `lock`, but user can interrupt the lock procedure anytime, and don’t have to worry about whether the lock exists or not.

Usage

dipsaus_lock(name, exclusive = TRUE, timeout = 10)
dipsaus_unlock(name, exclusive = TRUE, timeout = 10)

Arguments

name: character, the locker’s name, must be only letters and digits
exclusive: logical whether the locker is exclusive. True for write access, False for read access. Default is true.
timeout: numeric, seconds to wait for the locker to lock or unlock

Value

Logical, whether the operation succeed.

Examples

# unlock to prepare for the example
dipsaus_unlock('testlocker', timeout = 0.01)

# Create a locker, return TRUE
lock_success = dipsaus_lock('testlocker')
if(lock_success){
    cat2('testlocker has been locked')
}

# test whether locker has been locked
lock_success = dipsaus_lock('testlocker', timeout = 0.01)
if(!lock_success){
    cat2('attempt to lock testlocker failed')
}

# unlock
dipsaus_unlock('testlocker', timeout = 0.01)
**Description**

Asynchronous evaluator aims at queuing R evaluations from sub-processes without blocking the main session. It's based on 'parallel' and 'future' packages.

**Usage**

```r
make_async_evaluator(
  name,
  path = tempfile(),
  n_nodes = 1,
  n_subnodes = future::availableCores() - 1,
  verbose = FALSE,
  ...
)
```

**Arguments**

- `name`: unique name for the evaluator
- `path`: blank directory for evaluator to store data
- `n_nodes`: number of control nodes, default is 1
- `n_subnodes`: number of sub-sessions for each control node, default is the number of CPU cores minus 1
- `verbose`: for internal debug use
- `...`: passed to the constructor of `MasterEvaluator`

**Details**

'parallel' blocks the main session when evaluating expressions. 'future' blocks the main session when the number of running futures exceed the maximum number of workers. (For example if 4 workers are planned, then running 5 future instances at the same time will freeze the session).

Asynchronous evaluator is designed to queue any number of R expressions without blocking the main session. The incoming expressions are stored in `AbstractQueue` instances, and main session monitors the queue and is charge of notifying child sessions to evaluate these expressions whenever available.

Important: Asynchronous evaluator is not designed for super high-performance computing. The internal scheduler schedules `n_nodes` evaluations for every 1 second. Therefore if each of the process can be finished within $1 / n_nodes$ seconds, then use 'future' instead.
make_forked_clusters

Value

A list of functions to control the evaluator:

```
run(expr, success = NULL, failure = NULL, priority = 0, persist = FALSE, quoted = FALSE, ..., .list = NULL)
```

Queue and run an R expression.

- `expr` can be anything except for `q()`, which terminates the session. `rlang` `nse-force` is also supported. For example, you can use `!!` to quasi-quote the expression and unquote the values.
- `...`, `.list` provides additional data for `expr`. For example, `expr` uses a large data object `dat` in the main session, which might not be available to the child sessions. Also because the object is large, quasi-quotation could be slow or fail. By passing `dat=...` or `.list=list(dat=...)`, it's able to temporary store the data on hard-drive and persist for evaluators. The back-end is using `qs_map`, which is super fast for data that are no more than 2GB.

- `success` and `failure` functions to handle the results once the evaluator returns the value. Since it's almost impossible to know when the evaluator returns values, it's recommended that these functions to be simple.
- `priority` puts the priority of the expression. It can only be '0' or '1'. Evaluators will run expressions with priority equal to 1 first.
- `persist` indicates whether to run the expression and persist intermediate variables.

```
terminate()
```

Shut down and release all the resource.

```
scale_down(n_nodes, n_subnodes = 1), scale_up(n_nodes, n_subnodes = 1, create_if_missing = FALSE, path = tempfile())
```

Scale down or up the evaluator.

- `n_nodes` and `n_subnodes` see 'usage'
- `create_if_missing` If the evaluator was previously terminated or shutdown, setting this to be true ignores the 'invalid' flags and re-initialize the evaluator
- `path` If `create_if_missing` is true, then `path` will be passed to the constructor of `MasterEvaluator`.
  See 'usage'.

```
workers(...) Returns number of workers available in the evaluator. `...` is for debug use
```

```
progress() Returns a vector of 4 integers. They are:
```

1. The total number evaluations.
2. Number of running evaluations.
3. Number of awaiting evaluations.
4. Number of finished evaluations.
Usage

```r
make_forked_clusters(
  workers = future::availableCores(constraints = "multicore"),
  ...
)
```

Arguments

- `workers`: positive integer, number of cores to use
- `...`: passing to `future::plan`

Details

This is a wrapper for `future::plan(future::multicore,...)`. However, since version 1.14.0, forked clusters are disabled in 'RStudio' by default, and you usually need to enable it manually. This function provides a simple way of enable it and plan the future at the same time.

Value

- number of cores

---

**map**  
Create R object map.

---

Description

Provides five types of map that fit in different use cases.

Usage

```r
session_map(map = fastmap::fastmap())

rds_map(path = tempfile())

text_map(path = tempfile())

qs_map(path = tempfile())

redis_map(name = rand_string())
```

Arguments

- `map`: a `fastmap::fastmap()` list
- `path`: directory path where map data should be stored
- `name`: character, map name. If map names are the same, the data will be shared.
Details

There are five types of map implemented. They all inherit class AbstractMap. There are several differences in use case scenarios and they backend implementations.

*session_map* A session map takes a fastmap object. All objects are stored in current R session. This means you cannot access the map from other process nor parent process. The goal of this map is to share the data across different environments and to store global variables, as long as they share the same map object. If you are looking for maps that can be shared by different processes, check the rest map types. The closest map type is *redis_map*, which is also memory-based.

*rds_map* An 'RDS' map uses file system to store values. The values are stored separately in '.rds' files. Compared to session maps, 'RDS' map can be shared across different R process. It's recommended to store large files in rds_map. If the value is not large in RAM, text_map and redis_map are recommended.

*qs_map* A 'qs' map uses package 'qs' as backend. This map is very similar to rds_map, but is especially designed for large values. For example, pushing 1GB data to qs_map will be 100 times faster than using rds_map, and text_map will almost fail. However, compared to rds_map the stored data cannot be normally read by R as they are compressed binary files. And qs_map is heavier than text_map.

*text_map* A 'text' map uses file system to store values. Similar to rds_map, it can be stored across multiple processes as long as the maps share the same file directory. However, unlike rds_map, text_map the text_map can only store basic data values, namely atom data types. The supported types are: numeric, character, vector, list, matrix It's highly recommended to convert factors to characters. Do NOT use if the values are functions or environments. Please check write_yaml for details. The recommended use case scenario is when the speed is not the major concern, and you want to preserve data with backward compatibility. Otherwise it's highly recommended to use redis_map, qs_map, and rds_map.

*redis_map* A 'Redis' map uses free open source software 'Redis' and R package 'RcppRedis' as backend. Compared to session map, 'Redis' map can be shared across sessions. Compared to 'text' and 'rds' maps, 'Redis' map stores data in memory, meaning a potential of significant speed ups. To use redis_map, you need to install ‘Redis’ on your computer.

  - On Mac: use `brew install redis` to install and `brew services start redis` to start the service
  - On Linux: use `sudo apt-get install redis-server` to install and `sudo systemctl enable redis-server.service` to start the service
  - On Windows: Download from [https://github.com/dmajkic/redis/downloads](https://github.com/dmajkic/redis/downloads) and double click 'redis-server.exe'

Value

An R6 instance that inherits AbstractMap

Examples

```r
# ----------------------Basic Usage ----------------------

# Define a path to your map.
```
path = tempfile()
map <- qs_map(path)

# Reset
map$reset()

# Check if the map is corrupted.
map$validate()

# You have not set any key-value pairs yet.
# Let's say two parallel processes (A and B) are sharing this map.
# Process A set values
map$keys()

# Start push
# set a normal message
map$set(key = 'a', value = 1)

# set a large object
map$set(key = 'b', value = rnorm(100000))

# set an object with hash of another object
map$set(key = 'c', value = 2, signature = list(
  parameter1 = 123,
  parameter2 = 124
))

# Check what's in the map from process B
mapB <- qs_map(path)
mapB$keys()
mapB$keys(include_signatures = TRUE)

# Number of key-values pairs in the map.
mapB$size()

# Check if key exists
mapB$has(c('1','a','c'))

# Check if key exists and signature also matches
mapB$has('c', signature = list(
  parameter1 = 123,
  parameter2 = 124
))

# Signature changed, then return FALSE. This is especially useful when
# value is really large and reading the value takes tons of time
mapB$has('c', signature = list(
  parameter1 = 1244444,
  parameter2 = 124
))

# Destroy the map's files altogether.
mapB$destroy()
## Mask a function with given variables

**Description**

Modifies the default behavior of the function by adding one environment layer on top of input function. The masked variables are assigned directly to the environment.

**Usage**

```r
mask_function2(f, ..., .list = list())
```

**Arguments**

- `f`: any function
- `...`: .list: name-value pairs to mask the function

**Value**

a masked function

**Examples**

```r
a <- 123
f1 <- function(){
  a + 1
}
f1()  # 124

f2 <- mask_function2(f1, a = 1)
f2()  # a is masked with value 1, return 2

environment(f1)  # global env
environment(f2)  # masked env

env <- environment(f2)
identical(parent.env(env), environment(f1))  # true
env$a  # masked variables: a=1
```
MasterEvaluator  
*Generator Class for Asynchronous Evaluation*

Description

Generator Class for Asynchronous Evaluation

match_calls  
*Recursively match calls and modify arguments*

Description

Recursively match calls and modify arguments

Usage

```r
match_calls(
  call,
  recursive = TRUE,
  replace_args = list(),
  quoted = FALSE,
  envir = parent.frame(),
  ...
)
```

Arguments

call  
an R expression

recursive  
logical, recursively match calls, default is true

replace_args  
named list of functions, see examples

quoted  
logical, is call quoted

envir  
which environment should call be evaluated

...  
other parameters passing to `match.call`

Value

A nested call with all arguments matched
**Examples**

```r
library(dipsaus); library(shiny)

# In shiny modules, we might want to add ns() to inputIds
# In this example, textInput(id) will become textInput(ns(id))
match_calls(lapply(1:20, function(i){
  textInput(paste(\'\Var id\Var\', i), paste(\'\Var Label\Var\', i))
}), replace_args = list(
  inputId = function(arg, call){ as.call(list(quote(ns), arg)) })
)
```

---

**mem_limit2**

*Get max RAM size* This is an experimental function that is designed for non-windows systems

**Description**

Get max RAM size This is an experimental function that is designed for non-windows systems

**Usage**

```r
mem_limit2()
```

**Value**

a list of total free memory.

---

**new_function2**

*Create new function that supports 'quasi-quosure' syntax*

**Description**

Create new function that supports 'quasi-quosure' syntax

**Usage**

```r
new_function2(
  args = alist(),
  body = { },
  env = parent.frame(),
  quote_type = c("unquoted", "quote", "quo"),
  quasi_env = parent.frame()
)
```
Arguments

- **args**: named list of function formals
- **body**: function body expression, supports 'quasi-quosure' syntax
- **env**: declare environment of the function
- **quote_type**: character, whether body is unquoted, quoted, or a quo object
- **quasi_env**: where the 'quasi-quosure' should be evaluated, default is parent environment

Details

An unquoted body expression will be quoted, all the expressions with 'quasi-quosure' like `!!var` will be evaluated and substituted with the value of `var`. For a 'quosure', `quo_squash` will be applied. A quoted expression will not be substitute, but will be expanded if any 'quasi-quosure' detected.

- **args**: must be a list object, see **formals**. For arguments with no default values, or quoted defaults, use **alist**. An `arg=alist(a=)` will result in a function like `function(a){...}`. See examples for more details.

Value

- a function

See Also

- **quo**, **new_function**

Examples

```r
# ------------ standard usage ------------
x <- 1:10
f1 <- new_function2(alist(a=), { print(a + x) })
f1(0)

x <- 20:23
f1(0) # result changed as x changed

# ------------ 'quasi-quosure' syntax ------------
x <- 1:10
f2 <- new_function2(alist(a=), { print(a + !!x) })
print(f2)

f2(0)

x <- 20:23
f2(0) # result doesn't change as f2 doesn't depend on x anymore

# ------------ argument settings ------------
default <- 123
```
# default with values pre-specified
new_function2(list(a = default))  # function (a = 123)()

# default with values unevaluated
new_function2(list(a = quote(default)))  # function (a = default)()
new_function2(alist(a = default))

# missing default
new_function2(alist(a = ))  # function (a)()

no_op  

Pipe-friendly no-operation function

Description
returns the first input with side effects

Usage
no_op(.x, .expr, ..., .check_fun = TRUE)

Arguments
.x any R object
.expr R expression that produces side effects
..., .check_fun
see 'details'

Details
no_op is a pipe-friendly function that takes any values in, evaluate expressions but still returns input. This is very useful when you have the same input across multiple functions and you want to use pipes.

.expr is evaluated with a special object '.', you can use '.' to represent .x in .expr. For example, if .x=1:100, then plot(x=seq(0,1,length.out = 100),y=.) is equivalent to plot(x=seq(0,1,length.out = 100),y=1:100).

.check_fun checks whether .expr returns a function, if yes, then the function is called with argument .x and ...
Examples

library(magrittr)

## 1. Basic usage

# Will print(’a’) and return ’a’
no_op(’a’, print)

# Will do nothing and return ’a’ because .check_fun is false
no_op(’a’, print, .check_fun = FALSE)

# Will print(’a’) and return ’a’
no_op(’a’, print(.), .check_fun = FALSE)

## 2. Toy example

library(graphics)

par(mfrow = c(2,2))
x <- rnorm(100)

# hist and plot share the same input ’rnorm(100)’
x %>%
  no_op( hist, nclass = 10 ) %>%
  no_op( plot, x = seq(0,1,length.out = 100) ) %>%

# Repeat the previous two plots, but with different syntax
no_op({ hist(., nclass = 10) }) %>%
no_op({ plot(x = seq(0,1,length.out = 100), y = .) }) %>%

# The return statement is ignored
no_op({ return(x + 1)}) ->
y

# x is returned at the end
identical(x, y) # TRUE

---

**package_installed**  
_Check if a package is installed_

**Description**  
Check if a package is installed
Usage

package_installed(pkgs, all = FALSE)

Arguments

pkgs vector of package names
all only returns TRUE if all packages are installed. Default is FALSE.

Value

logical, if packages are installed or not. If all=TRUE, return a logical value of whether all packages are installed.

Examples

# Check if package base and dipsaus are installed
package_installed(c('base', 'dipsaus'))

# Check if all required packages are installed
package_installed(c('base', 'dipsaus'), all = TRUE)

parse_svec Parse Text Into Numeric Vectors (stable)

Description

Parse Text Into Numeric Vectors (stable)

Usage

parse_svec(text, sep = ',', connect = '-:|', sort = FALSE, unique = TRUE)

Arguments

text string with chunks, e.g. "1-10,14,16-20,18-30" has 4 chunks
sep default is ",", character used to separate chunks
connect characters defining connection links for example "1:10" is the same as "1-10"
sort sort the result
unique extract unique elements

Value

a numeric vector. For example, "1-3" returns c(1,2,3)
PersistContainer

See Also

deparse_svec

Examples

parse_svec('1-10, 13:15,14-20')

PersistContainer

Wrapper to cache key-value pairs and persist across sessions

Description

This class is designed to persist arbitrary R objects locally and share across different sessions. The container consists two-level caches. The first one is session-based, meaning it’s only valid under current R session and will be cleared once the session is shut down. The second is the persist-level map, which will persist to hard drive and shared across sessions. See cache method in ‘details’.

Public Methods

initialize(..., backend = text_map) The constructor. backend must inherit AbstractMap, ...
will be passed to backend$new(...). To check available back-ends and their use cases, see map.

reset(all = FALSE) Reset container. If all is set to be true, then reset session-based and hard-
drive-based, otherwise only reset session-based container.

destroy(all = FALSE) destroy the container. Only use it when you want to finalize the container
in reg.finalizer.

has(key, signature = NULL) returns a list of true/false (logical) vectors indicating whether keys
exist in the container, if signature is used when caching the key-value pairs, then it also checks
whether signature matches. This is very important as even if the keys match but signature is
wrong, the results will be false.

remove(keys, all = TRUE) Remove keys in the container. Default is to remove the keys in both
levels. If all=FALSE, then only remove the key in current session

cache(key, value, signature = NULL, replace = FALSE, persist = FALSE) key and signature
together form the unique identifier for the value. By default signature is none, but it’s very
useful when value if large, or key is not a string. replace indicates whether to force replace
the key-value pairs even if the entry exists. If persist is true, then the value is stored in
hard-disks, otherwise the value will be deleted once the session is closed.

See Also

map
Examples

```r
container = PersistContainer$new(tempfile())

# Reset the container so that values are cleared
container$reset(all = TRUE)

# Store '1' to 'a' with signature 111 to a non-persist map
# returns 1
container$cache(key = 'a', value = 1, signature = 111, persist = FALSE)

# Replace 'a' with 3
# returns 3
container$cache(key = 'a', value = 3, signature = 111,
               persist = TRUE, replace = TRUE)

# check if 'a' exists with signature 111
container$has('a', signature = 111) # TRUE
# When you only have 'a' but no signature
container$has('a') # TRUE
# check if 'a' exists with wrong signature 222
container$has('a', signature = 222) # FALSE

# Store 'a' with 2 with same signature
# will fail and ignore the value (value will not be evaluated if signatured)
# Return 2 (Important! use cached values)
container$cache(key = 'a', value = {
  print(123)
  return(2)
}, signature = 111, replace = FALSE)

# When no signature is present
# If the key exists (no signature provided), return stored value
# returns 3
container$cache(key = 'a', value = 4)

# replace is TRUE (no signature provided), signature will be some default value
container$cache(key = 'a', value = 2, replace = TRUE)

# destroy the container to free disk space
container$destroy()
```

**Description**

Register temporary code that will install packages at next session. The code will be automatically removed once executed.
Usage

prepare_install(
    packages,
    update_all = FALSE,
    restart = FALSE,
    repos = getOption("repos")
)

Arguments

packages characters, vector of package names
update_all whether to update all installed packages before installation; default is false
restart whether to restart session automatically
repos repositories to search for packages

Details

Installing packages in R session could require restarts if a package to be updated has been loaded. Normally restarting R fixes the problem. However, under some circumstances, such as with a startup code in profile, restarting R might still fail the installation. prepare_install inserts the installation code prior to the startup code so that next time the code will get executed before any other packages are loaded. Once the temporary code get executed, no matter succeeded or not, it will be removed from startup profile.

Value

None

progress2

'Shiny' progress bar, but can run without reactive context

Description

'Shiny' progress bar, but can run without reactive context

Usage

progress2(
    title,
    max = 1,
    ..., 
    quiet = FALSE,
    session = shiny::getDefaultReactiveDomain(),
    shiny_auto_close = FALSE,
    log = NULL
)
Arguments

- **title**: character, task description
- **max**: maximum number of items in the queue
- **...**: passed to `shiny::Progress$new(...)`
- **quiet**: suppress console output, ignored in shiny context.
- **session**: 'shiny' session, default is current reactive domain
- **shiny_auto_close**: logical, automatically close 'shiny' progress bar once current observer is over. Default is FALSE. If setting to TRUE, then it's equivalent to `p <- progress2(...); on.exit({p$close()},add = TRUE)`.
- **log**: function when running locally, default is NULL, which redirects to `cat2`

Value

A list of functions:

- `inc(detail, message = NULL, amount = 1, ...)`: Increase progress bar by amount (default is 1).
- `close()`: Close the progress
- `reset(detail = '', message = '', value = 0)`: Reset the progress to value (default is 0), and reset information
- `get_value()`: Get current progress value
- `is_closed()`: Returns logical value if the progress is closed or not.

Examples

```r
progress <- progress2('Task A', max = 2)
progress$inc('Detail 1')
progress$inc('Detail 2')
progress$close()

# Check if progress is closed
progress$is_closed()

# ------------------------------ Shiny Example ------------------------------
library(shiny)
library(dipsaus)

ui <- fluidPage(
  actionButtonStyled('do', 'Click Here', type = 'primary')
)

server <- function(input, output, session) {
  observeEvent(input$do, {
    updateActionButtonStyled(session, 'do', disabled = TRUE)
    progress <- progress2('Task A', max = 10, shiny_auto_close = TRUE)
    lapply(1:10, function(ii){
      progress$inc(sprintf('Detail %d', ii))
    })
  })
}
```
registerInputBinding

Register customized input to enable support by compound input

Description

Register customized input to enable support by compound input

Usage

registerInputBinding(fname, pkg, shiny_binding, update_function = NULL)

Arguments

fname character, function name, such as "textInput"
pkg character, package name, like "shiny"
shiny_binding character, 'JavaScript' binding name. See examples
update_function character, update function such as "shiny::textInput"

Value

a list of binding functions, one is 'JavaScript' object key in Shiny.inputBindings, the other is 'shiny' update function in R end.

Examples

# register shiny textInput
registerInputBinding('textInput', 'shiny',
  'shiny.textInput', 'shiny::updateTextInput')

# Register shiny actionLink
# In "Shiny.inputbindings", the binding name is "shiny.actionButtonInput",
# Shiny update function is "shiny::updateActionButton"
registerInputBinding('actionLink', 'shiny',
  'shiny.actionButtonInput', 'shiny::updateActionButton')
Description

Take a screenshot of the whole page and save encoded DataURI that can be accessed via input[[inputId]].

Usage

screenshot(inputId, session = shiny::getDefaultReactiveDomain())

Arguments

inputId the input id where the screenshot should be
session shiny session

Value

None. However, the screenshot results can be accessed from shiny input

Examples

library(shiny)
library(dipsaus)
ui <- fluidPage(
  tagList(
    shiny::singleton(shiny::tags$head(
      shiny::tags$link(rel="stylesheet", type="text/css", href="dipsaus/dipsaus.css"),
      shiny::tags$script(src="dipsaus/dipsaus-dipterix-lib.js")
    ))),
    actionButtonStyled('do', 'Take Screenshot'),
    compoundInput2('group', label = 'Group', components = list(
      shinystan::textInput('txt', 'Enter something here')
    ))
  )
)
server <- function(input, output, session) {
  observeEvent(input$do, {
    screenshot('screenshot_result')
  })
  observeEvent(input$screenshot_result, {
    showModal(modalDialog(
      tags$img(src = input$screenshot_result, width = '100%')
    ))
  })
}

if(interactive()){

set_shiny_input

**Set Shiny Input**

**Description**

Shiny ‘input’ object is read-only reactive list. When try to assign values to input, errors usually occur. This method provides several work-around to set values to input. Please use along with `use_shiny_dipsaus`.

**Usage**

```r
set_shiny_input(
  session = shiny::getDefaultReactiveDomain(),
  inputId,
  value,
  priority = c("event", "deferred", "immediate"),
  method = c("proxy", "serialize", "value", "expression"),
  quoted = TRUE
)
```

**Arguments**

- `session`: shiny session, see `shiny` domains
- `inputId`: character, input ID
- `value`: the value to assign
- `priority`: characters, options are "event", "deferred", and "immediate". "event" and "immediate" are similar, they always fire changes. "deferred" fire signals to other reactive/observers only when the input value has been changed
- `method`: characters, options are "proxy", "serialize", "value", "expression". "proxy" is recommended, other methods are experimental.
- `quoted`: is value quoted? Only used when method is "expression"

**Examples**

```r
library(shiny)
library(dipsaus)
ui <- fluidPage(
  # Register widgets
  use_shiny_dipsaus(),
  actionButton('run', 'Set Input'),
  verbatimTextOutput('input_value')
)
```
server <- function(input, output, session) {
  start = Sys.time()

  output$input_value <- renderPrint({
    now <- input$key
    now <- start
    cat('This app has been opened for ',
        difftime(now, start, units = 'sec'), ' seconds')
  })

  observeEvent(input$run, {
    # setting input$key to Sys.time()
    set_shiny_input(session, 'key', Sys.time())
  })
}

if(interactive()){
  shinyApp(ui, server)
}

shift_array(x, shift_idx, shift_by, shift_amount)

Arguments

x array, must have at least matrix
shift_idx which index is to be shifted
shift_by which dimension decides shift_amount
shift_amount shift amount along shift_idx

Details

A simple use-case for this function is to think of a matrix where each row is a signal and columns stand for time. The objective is to align (time-lock) each signal according to certain events. For each signal, we want to shift the time points by certain amount.

In this case, the shift amount is defined by shift_amount, whose length equals to number of signals. shift_idx=2 as we want to shift time points (column, the second dimension) for each signal. shift_by=1 because the shift amount is depend on the signal number.
Examples

```r
x <- matrix(1:10, nrow = 2, byrow = TRUE)
z <- shift_array(x, 2, 1, c(1,2))

y <- NA * x
y[1,1:4] = x[1,2:5]
y[2,1:3] = x[2,3:5]

# Check if z and y are the same
z - y

# array case
# x is Trial x Frequency x Time
x <- array(1:27, c(3,3,3))

# Shift time for each trial, amount is 1, -1, 0
shift_amount <- c(1,-1,0)
z <- shift_array(x, 3, 1, shift_amount)
par(mfrow = c(3, 2))
for( ii in 1:3 ){
  image(t(x[ii, ,]), ylab = 'Frequency', xlab = 'Time',
       main = paste('Trial', ii))
  image(t(z[ii, ,]), ylab = 'Frequency', xlab = 'Time',
       main = paste('Shifted amount:', shift_amount[ii]))
}
```

sync_shiny_inputs

Synchronize Shiny Inputs

Description

Synchronize Shiny Inputs

Usage

```r
sync_shiny_inputs(
  input,
  session,
  inputIds,
  uniform = rep("I", length(inputIds)),
  updates,
  snap = 250
)
```

Arguments

input, session  shiny reactive objects
inputIds  input ids to be synchronized
uniform  functions, equaling to length of inputIds, converting inputs to a uniform values
updates  functions, equaling to length of inputIds, updating input values
snap  numeric, milliseconds to defer the changes

Value
none.

Examples

library(shiny)

ui <- fluidPage(
  textInput('a', 'a', value = 'a'),
  sliderInput('b', 'b', value = 1, min = 0, max = 1000)
)

server <- function(input, output, session) {
  sync_shiny_inputs(input, session, inputIds = c('a', 'b'), uniform = list(
    function(a){as.numeric(a)},
    'I'
  ), updates = list(
    function(a){updateTextInput(session, 'a', value = a)},
    function(b){updateSliderInput(session, 'b', value = b)}
  ))
}

if( interactive() ){
  shinyApp(ui, server)
}

---

time_delta  

*Calculate time difference and return a number*

Description

Calculate time difference and return a number

Usage

time_delta(t1, t2, units = "secs")
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>time start</td>
</tr>
<tr>
<td>t2</td>
<td>time end</td>
</tr>
<tr>
<td>units</td>
<td>character, choices are 'secs', 'mins', 'hours', and 'days'</td>
</tr>
</tbody>
</table>

Value

numeric difference of time in units specified

Examples

```r
a = Sys.time()
Sys.sleep(0.3)
b = Sys.time()

time_delta(a, b) # In seconds, around 0.3
time_delta(a, b, 'mins') # in minutes, around 0.005
```

---

to_datauri

Convert file to 'base64' format

Description

Convert file to 'base64' format

Usage

```r
to_datauri(file, mime ="")
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>file path</td>
</tr>
<tr>
<td>mime</td>
<td>'mime' type, default is blank</td>
</tr>
</tbody>
</table>

Value

a 'base64' data string looks like 'data:;base64,AEF6986...'
to_ram_size  

Convert bytes to KB, MB, GB,....

Description
Convert bytes to KB, MB, GB,....

Usage
```r
to_ram_size(s, kb_to_b = 1000)
```

Arguments
- `s`: size
- `kb_to_b`: how many bytes counts one KB, 1000 by default

Value
numeric equaling to `s` but formatted

updateActionButtonStyled

Update styled action button

Description
Update styled action button

Usage
```r
updateActionButtonStyled(
  session,
  inputId,
  label = NULL,
  icon = NULL,
  type = NULL,
  disabled = NULL,
  ...
)
```

Arguments
- `session`, `inputId`, `label`, `icon` passed to `shiny::updateActionButton`
- `type`: button type to update
- `disabled`: whether to disable the button
- `...`: ignored
Value

none

See Also

`actionButtonStyled` for how to define the button.

updateCompoundInput2  

Update compound inputs

Description

Update compound inputs

Usage

updateCompoundInput2(
  session,
  inputId,
  value = NULL,
  ncomp = NULL,
  initialization = NULL,
  ...
)

Arguments

  session    shiny session or session proxy
  inputId    character see `compoundInput2`
  value      list of lists, see `compoundInput2` or examples
  ncomp      integer, non-negative number of groups to update, NULL to remain unchanged
  initialization,...
            named list of other updates

Value

none

See Also

`compoundInput2` for how to define components.
**update_fastmap2**

*Migrate a fastmap2 object to a new one*

**Description**

Migrate a fastmap2 object to a new one

**Usage**

```r
update_fastmap2(from, to, override = TRUE)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>from, to</td>
<td>fastmap2 object</td>
</tr>
<tr>
<td>override</td>
<td>whether to override keys in to if they exist</td>
</tr>
</tbody>
</table>
Value

Map to

See Also

fastmap2

---

use_shiny_dipsaus  Set up shiny plugins

### Description

This function must be called from a Shiny app’s UI in order for some widgets to work.

### Usage

```r
use_shiny_dipsaus()
```

---

%=>%  A JavaScript style of creating functions

### Description

A JavaScript style of creating functions

### Usage

```r
args %=>% expr
```

#### Arguments

<table>
<thead>
<tr>
<th>args</th>
<th>function arguments: see <code>formals</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>R expression that forms the body of functions: see <code>body</code></td>
</tr>
</tbody>
</table>

### Value

A function that takes `args` as parameters and `expr` as the function body
Examples

# Formal arguments
c(a) %>% {
  print(a)
}

# Informal arguments
list(a=) %>% {
  print(a)
}

# Multiple inputs
c(a, b = 2, ...) %>% {
  print(c(a, b, ...))
}

# ----- JavaScript style of forEach -----  
# ### Equivalent JavaScript Code:
# LETTERS.forEach((el, ii) => {
#   console.log('The index of letter ' + el + ' in "x" is: ' + ii);
# });

iapply(LETTERS, c(el, ii) %>% {
  cat2('The index of letter ', el, ' in ', sQuote('x'), ' is: ', ii)
}) -> results

---

%?<-%

Assign if not exists, or NULL Provides a way to assign default values to variables. If the statement `lhs` is invalid or NULL, this function will try to assign value, otherwise nothing happens.

Description

Assign if not exists, or NULL Provides a way to assign default values to variables. If the statement `lhs` is invalid or NULL, this function will try to assign value, otherwise nothing happens.

Usage

`lhs %?<-% value`

Arguments

- `lhs`: an object to check or assign value
- `value`: value to be assigned if `lhs` is NULL

Value

Assign value on the right-hand side to the left-hand side if `lhs` does not exist or is NULL
Examples

# Prepare, remove aaa if exists
if(exists('aaa', envir = globalenv(), inherits = FALSE)){
  rm(aaa, envir = globalenv())
}

# Assign
aaa %?-% 1; print(aaa)

# However, if assigned, nothing happens
aaa = 1;
aaa %?-% 2;
print(aaa)

# in a list
a = list()
a$e %?-% 1; print(a$e)
a$e %?-% 2; print(a$e)

%+-%

Plus-minus operator

Description

Plus-minus operator

Usage

a %+-% b

Arguments

a, b numeric vectors, matrices or arrays

Value

a +/-b, the dimension depends on a+b. If a+b is a scalar, returns a vector of two; in the case of vector, returns a matrix; all other cases will return an array with the last dimension equal to 2.

Examples

# scalar
1 %+-% 2 # -1, 3

# vector input
c(1,2,3) %+-% 2 # matrix
# matrix input
matrix(1:9, 3) %+-% 2  # 3x3x2 array
Index

[.fastmap2 (fastmap2), 26
[<- .fastmap2 (fastmap2), 26
[[.fastmap2 (fastmap2), 26
[[<- .fastmap2 (fastmap2), 26
$.fastmap2 (fastmap2), 26
$<- .fastmap2 (fastmap2), 26
%D (decorate_function), 20
%+%, 66
%=>%, 64
%?-%, 65

AbstractMap, 3, 41
AbstractQueue, 3, 38
actionButtonStyled, 6, 62
adjustcolor, 15, 16
aggregate, 23, 24
alist, 46
as.list.fastmap2 (fastmap2), 26
ask_or_default, 7, 8
ask_yesno, 8, 8
async, 9
async_expr, 9, 9
async_flapply, 10

base64_to_image, 11
drop_nulls, 24
drop_nulls, 24
drop_nulls, 24
drop_nulls, 24
drop_nulls, 24
drop_nulls, 24
drop_nulls, 24
drop_nulls, 24

drop_nulls, 24
drop_nulls, 24
drop_nulls, 24
drop_nulls, 24
drop_nulls, 24
drop_nulls, 24

 Hasan, 68

domains, 56
drop_nulls, 24
eval_dirty, 25

fastmap, 26, 41
fastmap2, 26, 64
force, 31
forelse, 28
formals, 46, 64
future_eapply, 35
future_lapply, 10

get_cpu, 30
get_dots, 31, 31
get_ram, 32
getInputBinding, 29

handler_dipsaus_progress, 32

iapply, 34
interactive, 7, 8

lapply, 34
lapply_async2, 35


length.fastmap2 (fastmap2), 26
list_to_fastmap2, 36
lock, 37, 37

make_async_evaluator, 38
make_forked_clusters, 39
make_progression_handler, 33

map, 40, 50
mask_function2, 43
MasterEvaluator, 38, 39, 44

match_calls, 44
mem_limit2, 45

names.fastmap2 (fastmap2), 26
new_function, 46
new_function2, 45

68
no_op, 47
package_installed, 48
parse_svec, 22, 49
PersistContainer, 50
plan, 35
prepare_install, 51
print.fastmap2 (fastmap2), 26
progress2, 32, 33, 52
qs_map, 39, 41
qs_map (map), 40
quo, 46
quo_squash, 46
rds_map, 41
rds_map (map), 40
readline, 8
redis_map, 41
redis_map (map), 40
reg.finalizer, 50
registerInputBinding, 54
sapply, 34
screenshot, 55
session_map, 41
session_map (map), 40
set_shiny_input, 56
shift_array, 57
symbol, 14
sync_shiny_inputs, 58
text_map, 41
text_map (map), 40
time_delta, 59
to_datauri, 60
to_ram_size, 61
update_fastmap2, 63
updateActionButtonStyled, 6, 61
updateCompoundInput2, 78, 62
use_shiny_dipsaus, 56, 64
write_yaml, 41