Package 'async'

May 31, 2023

Type Package

Title Async

Version 0.0.6

Maintainer Meantrix <dev@meantrix.com>

Description A simple tools for passing messages between some R processes like shiny reactives and futures routines.

Depends R (>= 3.6.0),
shiny (>= 1.4.0),
R6 (>= 2.4.1),
checkmate,
stringi,
shinyjs,
magrittr,
uuid

License MIT License

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

R topics documented:

async *R6 Class async*

Description

simple tool to pass message of progress and interruption of routines between some R processes like shiny reactives and futures routines.

Active bindings

`status_file` path to temporary file

`upper` The value that represents the end of the routine progress.

`lower` The value that represents the starting point of the routine progress.

`auto.finish` if it is true the routine will be interrupted when the progress is equal to the upper value.

`value_message` get job status.

`reactive` If its true generate a reactive expression is a expression whose result will change over time.

Methods

Public methods:

- `async$new()`
- `async$interrupt()`
- `async$set()`
- `async$inc()`
- `async$status()`
- `async$check()`
- `async$finalize()`
- `async$clone()`

Method `new()`**:** create an interactive environment to pass progress message between R processes.

Usage:

```
async$new(
  lower,
  upper,
  auto.finish = TRUE,
  reactive = TRUE,
  verbose = FALSE,
  msg,
  detail
)
```

Arguments:

`lower` The value that represents the starting point of the routine progress.

`upper` upper The value that represents the end of the routine progress.

`auto.finish` auto.finish if it is true the routine will be interrupted when the progress is equal to the upper value.

`reactive` If its true generate a reactive expression is a expression whose result will change over time.

`verbose` logical.If its TRUE provides much more information about the flow of information between the R processes.

`msg` A single-element character vector; the initial message to be pass between processes.

`detail` A single-element character vector; the initial detail message to be pass between processes.

Method `interrupt()`**:** Interrupt the process tracked by the async function.

Usage:

```
async$interrupt(msg = "process interrupted", detail = "")
```

Arguments:

`msg` A single-element character vector; the interrupt message to be pass between processes.

`detail` A single-element character vector; the interrupt detail message to be pass between processes.

Method `set()`**:** Set progress of the tracked routine.

Usage:

```
async$set(value = 0, msg = "Running...", detail = "")
```

Arguments:

`value` the value at which to set the progress, relative to lower and upper.

`msg` A single-element character vector; the message to be pass between processes.

`detail` A single-element character vector; the detail message to be pass between processes.

Method `inc()`**:** Increment the progress of the tracked routine.

Usage:

```
async$inc(value = 0, msg = "Running...", detail = "")
```

Arguments:

`value` the value at which to set the progress, relative to lower and upper.

`msg` A single-element character vector; the message to be pass between processes.

`detail` A single-element character vector; the detail message to be pass between processes.

Method `status()`**:** get the status of progress out of the context being tracked.

Usage:

```
async$status()
```

Method `check()`**:** Check the status process in the tracked context

Usage:

```
async$check()
```

Method `finalize()`**:** Close all routines of async tracking.

Usage:

```
async$finalize()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
async$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`asyncBar1` *R6 Class asyncBar1*

Description

Reports async class progress with a default shiny bar to end user.

Methods

Public methods:

- `asyncBar1$new()`
- `asyncBar1$progress()`
- `asyncBar1$finalize()`
- `asyncBar1$clone()`

Method `new()`**:** Interactive environment to create a progress bar of async tracking.

Usage:

```
asyncBar1$new(async, id, interval = 1000, max.rep = 50)
```

Arguments:

`async` R6 class. Object to tracking routines.

`id` character. ID of progress bar.

`interval` numeric. Approximate number of milliseconds to wait between checks of the job progress.

`max.rep` numeric. Maximum number of times that a progress value can be repeated.

Method `progress()`**:** observe event to checks of the async job progress.

Usage:

```
asyncBar1$progress(session, input)
```

Arguments:

`session` shiny session

`input` shiny input

Method `finalize()`**:** Close all routines of async bar.

Usage:

```
asyncBar1$finalize()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
asyncBar1$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`asyncBar2` *R6 Class asyncBar1*

Description

Reports async class progress with a custom shiny bar to end user.

Methods

Public methods:

- `asyncBar2$new()`
- `asyncBar2$progress()`
- `asyncBar2$cancel()`
- `asyncBar2$finalize()`
- `asyncBar2$clone()`

Method `new()`**:** Interactive environment to create a progress bar of async tracking.

Usage:

```
asyncBar2$new(async, id, interval = 1000, max.rep = 50, msg)
```

Arguments:

`async` R6 class. Object to tracking routines.

`id` character. ID of progress bar.

`interval` numeric. Approximate number of milliseconds to wait between checks of the job progress.

`max.rep` numeric. Maximum number of times that a progress value can be repeated.

`msg` A single-element character vector; the message to be displayed to the user,

Method `progress()`**:** observe event to checks of the async job progress.

Usage:

```
asyncBar2$progress(session, input)
```

Arguments:

`session` shiny session

`input` shiny input

Method `cancel()`: display a cancel button to end user associated with the process tracked by the async class

Usage:

```
asyncBar2$cancel(session, input, not.msg)
```

Arguments:

`session` shiny session

`input` shiny input

`not.msg` Content of notification message

Method `finalize()`: Close all routines of async bar.

Usage:

```
asyncBar2$finalize()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
asyncBar2$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`hello` *Hello, World!*

Description

Prints 'Hello, world!'.

Usage

```
hello()
```

Examples

```
hello()
```

`load_async` *save_async*

Description

Function to read a single async object saved as rds.

Usage

```
load_async(file)
```

Arguments

`file`	a connection or the name of the file where the R object is saved to or read from.

`reactiveTrigger` *reactiveTrigger*

Description

A reactive trigger can be used when you want to be able to explicitly trigger a reactive expression. You can think of it as being similar to an action button, except instead of clicking on a button to trigger an expression, you can programatically cause the trigger. This concept and code was created by Joe Cheng (author of shiny).

Usage

```
reactiveTrigger()
```

`save_async` *save_async*

Description

Functions to write a single async object to a RDS file.

Usage

```
save_async(async, file)
```

Arguments

`async`	R6 class. Object to tracking routines.
`file`	a connection or the name of the file where the R object is saved to or read from. If its is missing a unique name file is created by default.

Index