Package 'later'

February 11, 2019

Type Package

Title Utilities for Delaying Function Execution

Version 0.8.0

Description Executes arbitrary R or C functions some time after the current time, after the R execution stack has emptied.

URL https://github.com/r-lib/later

BugReports https://github.com/r-lib/later/issues

License GPL (>= 2)

Imports Rcpp (>= 0.12.9), rlang

LinkingTo Rcpp, BH

RoxygenNote 6.1.0

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

NeedsCompilation yes

Author Joe Cheng [aut, cre],
RStudio [cph],
Winston Chang [aut],
Marcus Geelnard [ctb, cph] (TinyCThread library,
https://tinycthread.github.io/),
Evan Nemerson [ctb, cph] (TinyCThread library,
https://tinycthread.github.io/)

Maintainer Joe Cheng <joe@rstudio.com>

Repository CRAN

Date/Publication 2019-02-11 08:53:37 UTC

R topics documented:

later *Executes a function later*

Description

Schedule an R function or formula to run after a specified period of time. Similar to JavaScript's `setTimeout` function. Like JavaScript, R is single-threaded so there's no guarantee that the operation will run exactly at the requested time, only that at least that much time will elapse.

Usage

```
later(func, delay = 0)
```

Arguments

`func`	A function or formula (see `rlang::as_function()`).
`delay`	Number of seconds in the future to delay execution. There is no guarantee that the function will be executed at the desired time, but it should not execute earlier.

Details

The mechanism used by this package is inspired by Simon Urbanek's background package and similar code in Rhttpd.

Note

To avoid bugs due to reentrancy, by default, scheduled operations only run when there is no other R code present on the execution stack; i.e., when R is sitting at the top-level prompt. You can force past-due operations to run at a time of your choosing by calling `run_now()`.

Error handling is not particularly well-defined and may change in the future. options(error=browser) should work and errors in `func` should generally not crash the R process, but not much else can be said about it at this point. If you must have specific behavior occur in the face of errors, put error handling logic inside of `func`.

Examples

```
# Example of formula style
later(~cat("Hello from the past\n"), 3)

# Example of function style
later(function() {
  print(summary(cars))
}, 2)
```

next_op_secs *Relative time to next scheduled operation*

Description

Returns the duration between now and the earliest operation that is currently scheduled, in seconds. If the operation is in the past, the value will be negative. If no operation is currently scheduled, the value will be `Inf`.

Usage

```
next_op_secs()
```

run_now *Execute scheduled operations*

Description

Normally, operations scheduled with `later()` will not execute unless/until no other R code is on the stack (i.e. at the top-level). If you need to run blocking R code for a long time and want to allow scheduled operations to run at well-defined points of your own operation, you can call `run_now()` at those points and any operations that are due to run will do so.

Usage

```
run_now(timeoutSecs = 0L, all = TRUE)
```

Arguments

`timeoutSecs`	Wait (block) for up to this number of seconds waiting for an operation to be ready to run. If `0`, then return immediately if there are no operations that are ready to run. If `Inf` or negative, then wait as long as it takes (if none are scheduled, then this will block forever).
`all`	If `FALSE`, `run_now()` will execute at most one scheduled operation (instead of all eligible operations). This can be useful in cases where you want to interleave scheduled operations with your own logic.

Details

If one of the callbacks throws an error, the error will *not* be caught, and subsequent callbacks will not be executed (until `run_now()` is called again, or control returns to the R prompt). You must use your own tryCatch if you want to handle errors.

Value

A logical indicating whether any callbacks were actually run.

Index