

Package ‘jobqueue’

February 22, 2025

Type Package

Title Run Interruptible Code Asynchronously

Version 1.4.1

Date 2025-02-21

Description Takes an R expression and returns a Job object with a \$stop() method which can be called to terminate the background job. Also provides timeouts and other mechanisms for automatically terminating a background job. The result of the expression is available synchronously via \$result or asynchronously with callbacks or through the 'promises' package framework.

URL <https://cmmr.github.io/jobqueue/>, <https://github.com/cmmr/jobqueue>

BugReports <https://github.com/cmmr/jobqueue/issues>

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

Config/Needs/website rmarkdown

Config/testthat/edition 3

Config/testthat/parallel false

Depends R (>= 4.2.0)

Imports cli, later, magrittr, parallelly, promises, ps, R6, rlang, semaphore, utils

Suggests glue, knitr, rmarkdown, testthat (>= 3.0.0)

NeedsCompilation no

Author Daniel P. Smith [aut, cre] (<<https://orcid.org/0000-0002-2479-2044>>), Alkek Center for Metagenomics and Microbiome Research [cph, fnd]

Maintainer Daniel P. Smith <dansmith01@gmail.com>

Repository CRAN

Date/Publication 2025-02-21 23:20:02 UTC

Contents

Job	2
Queue	5
Worker	9
Index	13

Job	<i>How to Evaluate an R Expression</i>
-----	--

Description

The Job object encapsulates an expression and its evaluation parameters. It also provides a way to check for and retrieve the result.

Active bindings

`expr` R expression that will be run by this Job.

`vars` Get or set - List of variables that will be placed into the expression's environment before evaluation.

`reformat` Get or set - function (`job`) for defining `<Job>$result`.

`signal` Get or set - Conditions to signal.

`cpus` Get or set - Number of CPUs to reserve for evaluating `expr`.

`timeout` Get or set - Time limits to apply to this Job.

`proxy` Get or set - Job to proxy in place of running `expr`.

`state` Get or set - The Job's state: 'created', 'submitted', 'queued', 'dispatched', 'starting', 'running', or 'done'. *Assigning to `<Job>$state` will trigger callback hooks.*

`output` Get or set - Job's raw output. *Assigning to `<Job>$output` will change the Job's state to 'done'.*

`result` Result of `expr`. Will block until Job is finished.

`hooks` Currently registered callback hooks as a named list of functions. Set new hooks with `<Job>$on()`.

`is_done` TRUE or FALSE depending on if the Job's result is ready.

`uid` A short string, e.g. 'J16', that uniquely identifies this Job.

Methods

Public methods:

- `Job$new()`
- `Job$print()`
- `Job$on()`
- `Job$wait()`
- `Job$stop()`

Method `new()`: Creates a Job object defining how to run an expression on a background worker process.

Typically you won't need to call `Job$new()`. Instead, create a [Queue](#) and use `<Queue>$run()` to generate Job objects.

Usage:

```
Job$new(
  expr,
  vars = NULL,
  timeout = NULL,
  hooks = NULL,
  reformat = NULL,
  signal = FALSE,
  cpus = 1L,
  ...
)
```

Arguments:

`expr` A call or R expression wrapped in curly braces to evaluate on a worker. Will have access to any variables defined by `vars`, as well as the Worker's globals, packages, and init configuration. See `vignette('eval')`.

`vars` A named list of variables to make available to `expr` during evaluation. Alternatively, an object that can be coerced to a named list with `as.list()`, e.g. named vector, data.frame, or environment. Or a function (`job`) that returns such an object.

`timeout` A named numeric vector indicating the maximum number of seconds allowed for each state the job passes through, or 'total' to apply a single timeout from 'submitted' to 'done'. Or a function (`job`) that returns the same. Example: `timeout = c(total = 2.5, running = 1)`. See `vignette('stops')`.

`hooks` A named list of functions to run when the Job state changes, of the form `hooks = list(created = function (worker) {...})`. Or a function (`job`) that returns the same. Names of worker hooks are typically 'created', 'submitted', 'queued', 'dispatched', 'starting', 'running', 'done', or '*' (duplicates okay). See `vignette('hooks')`.

`reformat` Set `reformat = function (job)` to define what `<Job>$result` should return. The default, `reformat = NULL` passes `<Job>$output` to `<Job>$result` unchanged. See `vignette('results')`.

`signal` Should calling `<Job>$result` signal on condition objects? When FALSE, `<Job>$result` will return the object without taking additional action. Setting to TRUE or a character vector of condition classes, e.g. `c('interrupt', 'error', 'warning')`, will cause the equivalent of `stop(<condition>)` to be called when those conditions are produced. Alternatively, a function (`job`) that returns TRUE or FALSE. See `vignette('results')`.

`cpus` How many CPU cores to reserve for this Job. Or a function (`job`) that returns the same. Used to limit the number of Jobs running simultaneously to respect `<Queue>$max_cpus`. Does not prevent a Job from using more CPUs than reserved.

... Arbitrary named values to add to the returned Job object.

Returns: A Job object.

Method `print()`: Print method for a Job.

Usage:

`Job#print(...)`

Arguments:

... Arguments are not used currently.

Returns: This Job, invisibly.

Method `on()`: Attach a callback function to execute when the Job enters state.

Usage:

`Job$on(state, func)`

Arguments:

`state` The name of a Job state. Typically one of:

- '*' - Every time the state changes.
- '.next' - Only one time, the next time the state changes.
- 'created' - After `Job$new()` initialization.
- 'submitted' - After `<Job>$queue` is assigned.
- 'queued' - After `stop_id` and `copy_id` are resolved.
- 'dispatched' - After `<Job>$worker` is assigned.
- 'starting' - Before evaluation begins.
- 'running' - After evaluation begins.
- 'done' - After `<Job>$output` is assigned.

Custom states can also be specified.

`func` A function that accepts a Job object as input. You can call `<Job>$stop()` or edit `<Job>$values` and the changes will be persisted (since Jobs are reference class objects). You can also edit/stop other queued jobs by modifying the Jobs in `<Job>$queue$jobs`. Return value is ignored.

Returns: A function that when called removes this callback from the Job.

Method `wait()`: Blocks until the Job enters the given state.

Usage:

`Job$wait(state = "done", timeout = NULL)`

Arguments:

`state` The name of a Job state. Typically one of:

- '*' - Every time the state changes.
- '.next' - Only one time, the next time the state changes.
- 'created' - After `Job$new()` initialization.
- 'submitted' - After `<Job>$queue` is assigned.
- 'queued' - After `stop_id` and `copy_id` are resolved.
- 'dispatched' - After `<Job>$worker` is assigned.
- 'starting' - Before evaluation begins.
- 'running' - After evaluation begins.
- 'done' - After `<Job>$output` is assigned.

Custom states can also be specified.

`timeout` Stop the Job if it takes longer than this number of seconds, or NULL.

Returns: This Job, invisibly.

Method stop(): Stop this Job. If the Job is running, its Worker will be restarted.

Usage:

```
Job$stop(reason = "job stopped by user", cls = NULL)
```

Arguments:

reason A message to include in the 'interrupt' condition object that will be returned as the Job's result. Or a condition object.

cls Character vector of additional classes to prepend to c('interrupt', 'condition').

Returns: This Job, invisibly.

Queue

Assigns Jobs to a Set of Workers

Description

Jobs go in. Results come out.

Active bindings

hooks A named list of currently registered callback hooks.

jobs Get or set - List of [Jobs](#) currently managed by this Queue.

state The Queue's state: 'starting', 'idle', 'busy', 'stopped', or 'error.'

uid Get or set - Unique identifier, e.g. 'Q1'.

tmp The Queue's temporary directory.

workers Get or set - List of [Workers](#) used for processing Jobs.

cmd The error that caused the Queue to stop.

Methods

Public methods:

- [Queue\\$new\(\)](#)
- [Queue\\$print\(\)](#)
- [Queue\\$run\(\)](#)
- [Queue\\$submit\(\)](#)
- [Queue\\$wait\(\)](#)
- [Queue\\$on\(\)](#)
- [Queue\\$stop\(\)](#)

Method new(): Creates a pool of background processes for handling \$run() and \$submit() calls. These workers are initialized according to the globals, packages, and init arguments.

Usage:

```
Queue$new(
  globals = NULL,
  packages = NULL,
  init = NULL,
  max_cpus = availableCores(),
  workers = ceiling(max_cpus * 1.2),
  timeout = NULL,
  hooks = NULL,
  reformat = NULL,
  signal = FALSE,
  cpus = 1L,
  stop_id = NULL,
  copy_id = NULL
)
```

Arguments:

globals A named list of variables that all `<Job>$exprs` will have access to. Alternatively, an object that can be coerced to a named list with `as.list()`, e.g. named vector, `data.frame`, or environment.

packages Character vector of package names to load on workers.

init A call or R expression wrapped in curly braces to evaluate on each worker just once, immediately after start-up. Will have access to variables defined by `globals` and assets from `packages`. Returned value is ignored.

max_cpus Total number of CPU cores that can be reserved by all running Jobs (`sum(<Job>$cpus)`). Does not enforce limits on actual CPU utilization.

workers How many background [Worker](#) processes to start. Set to more than `max_cpus` to enable standby Workers to quickly swap out with Workers that need to restart.

timeout, hooks, reformat, signal, cpus, stop_id, copy_id Defaults for this Queue's `$run()` method. Here only, `stop_id` and `copy_id` must be either a function (`job`) or `NULL`. `hooks` can set queue, worker, and/or job hooks - see the "Attaching" section in `vignette('hooks')`.

Returns: A Queue object.

Method `print()`: Print method for a Queue.

Usage:

```
Queue$print(...)
```

Arguments:

... Arguments are not used currently.

Method `run()`: Creates a Job object and submits it to the queue for running. Any NA arguments will be replaced with their value from `Queue$new()`.

Usage:

```
Queue$run(
  expr,
  vars = list(),
  timeout = NA,
  hooks = NA,
  reformat = NA,
```

```

    signal = NA,
    cpus = NA,
    stop_id = NA,
    copy_id = NA,
    ...
  )

```

Arguments:

- expr** A call or R expression wrapped in curly braces to evaluate on a worker. Will have access to any variables defined by **vars**, as well as the Worker's globals, packages, and init configuration. See `vignette('eval')`.
- vars** A named list of variables to make available to **expr** during evaluation. Alternatively, an object that can be coerced to a named list with `as.list()`, e.g. named vector, data.frame, or environment. Or a function `(job)` that returns such an object.
- timeout** A named numeric vector indicating the maximum number of seconds allowed for each state the job passes through, or 'total' to apply a single timeout from 'submitted' to 'done'. Can also limit the 'starting' state for Workers. A function `(job)` can be used in place of a number. Example: `timeout = c(total = 2.5, running = 1)`. See `vignette('stops')`.
- hooks** A named list of functions to run when the Job state changes, of the form `hooks = list(created = function(worker) {...})`. Or a function `(job)` that returns the same. Names of worker hooks are typically 'created', 'submitted', 'queued', 'dispatched', 'starting', 'running', 'done', or '*' (duplicates okay). See `vignette('hooks')`.
- reformat** Set `reformat = function(job)` to define what `<Job>$result` should return. The default, `reformat = NULL` passes `<Job>$output` to `<Job>$result` unchanged. See `vignette('results')`.
- signal** Should calling `<Job>$result` signal on condition objects? When FALSE, `<Job>$result` will return the object without taking additional action. Setting to TRUE or a character vector of condition classes, e.g. `c('interrupt', 'error', 'warning')`, will cause the equivalent of `stop(<condition>)` to be called when those conditions are produced. Alternatively, a function `(job)` that returns TRUE or FALSE. See `vignette('results')`.
- cpus** How many CPU cores to reserve for this Job. Or a function `(job)` that returns the same. Used to limit the number of Jobs running simultaneously to respect `<Queue>$max_cpus`. Does not prevent a Job from using more CPUs than reserved.
- stop_id** If an existing Job in the Queue has the same `stop_id`, that Job will be stopped and return an 'interrupt' condition object as its result. `stop_id` can also be a function `(job)` that returns the `stop_id` to assign to a given Job. A `stop_id` of NULL disables this feature. See `vignette('stops')`.
- copy_id** If an existing Job in the Queue has the same `copy_id`, the newly submitted Job will become a "proxy" for that earlier Job, returning whatever result the earlier Job returns. `copy_id` can also be a function `(job)` that returns the `copy_id` to assign to a given Job. A `copy_id` of NULL disables this feature. See `vignette('stops')`.
- ...** Arbitrary named values to add to the returned Job object.

Returns: The new Job object.

Method `submit()`: Adds a Job to the Queue for running on a background process.

Usage:

```
Queue$submit(job)
```

Arguments:

job A `Job` object, as created by `Job$new()`.

Returns: This Queue, invisibly.

Method `wait()`: Blocks until the Queue enters the given state.

Usage:

```
Queue$wait(state = "idle", timeout = NULL, signal = TRUE)
```

Arguments:

state The name of a Queue state. Typically one of:

- '*' - Every time the state changes.
- '.next' - Only one time, the next time the state changes.
- 'starting' - Workers are starting.
- 'idle' - All workers are ready/idle.
- 'busy' - At least one worker is busy.
- 'stopped' - Shutdown is complete.

timeout Stop the Queue if it takes longer than this number of seconds, or NULL.

signal Raise an error if encountered (will also be recorded in `<Queue>$cnd`).

Returns: This Queue, invisibly.

Method `on()`: Attach a callback function to execute when the Queue enters state.

Usage:

```
Queue$on(state, func)
```

Arguments:

state The name of a Queue state. Typically one of:

- '*' - Every time the state changes.
- '.next' - Only one time, the next time the state changes.
- 'starting' - Workers are starting.
- 'idle' - All workers are ready/idle.
- 'busy' - At least one worker is busy.
- 'stopped' - Shutdown is complete.

func A function that accepts a Queue object as input. Return value is ignored.

Returns: A function that when called removes this callback from the Queue.

Method `stop()`: Stop all jobs and workers.

Usage:

```
Queue$stop(reason = "job queue shut down by user", cls = NULL)
```

Arguments:

reason Passed to `<Job>$stop()` for any Jobs currently managed by this Queue.

cls Passed to `<Job>$stop()` for any Jobs currently managed by this Queue.

Returns: This Queue, invisibly.

Worker *A Background Process*

Description

Where [Job](#) expressions are evaluated.

Active bindings

`hooks` A named list of currently registered callback hooks.

`job` The currently running Job.

`ps` The `ps::ps_handle()` object for the background process.

`state` The Worker's state: 'starting', 'idle', 'busy', or 'stopped'.

`uid` A short string, e.g. 'W11', that uniquely identifies this Worker.

`tmp` The Worker's temporary directory.

`cmd` The error that caused the Worker to stop.

Methods

Public methods:

- [Worker\\$new\(\)](#)
- [Worker\\$print\(\)](#)
- [Worker\\$start\(\)](#)
- [Worker\\$stop\(\)](#)
- [Worker\\$restart\(\)](#)
- [Worker\\$on\(\)](#)
- [Worker\\$wait\(\)](#)
- [Worker\\$run\(\)](#)

Method `new()`: Creates a background R process for running [Jobs](#).

Usage:

```
Worker$new(
  globals = NULL,
  packages = NULL,
  init = NULL,
  hooks = NULL,
  wait = TRUE,
  timeout = Inf
)
```

Arguments:

`globals` A named list of variables that all `<Job>$exprs` will have access to. Alternatively, an object that can be coerced to a named list with `as.list()`, e.g. named vector, `data.frame`, or environment.

packages Character vector of package names to load on workers.

init A call or R expression wrapped in curly braces to evaluate on each worker just once, immediately after start-up. Will have access to variables defined by `globals` and `assets` from packages. Returned value is ignored.

hooks A named list of functions to run when the Worker state changes, of the form `hooks = list(idle = function (worker) { . . . })`. Names of worker hooks are typically `starting`, `idle`, `busy`, `stopped`, or `'*'` (duplicates okay). See `vignette('hooks')`.

wait If TRUE, blocks until the Worker is 'idle'. If FALSE, the Worker object is returned in the 'starting' state.

timeout How long to wait for the worker to finish starting (in seconds). If NA, defaults to the `Worker$new()` argument.

Returns: A Worker object.

Method `print()`: Print method for a Worker.

Usage:

```
Worker$print(...)
```

Arguments:

. . . Arguments are not used currently.

Returns: The Worker, invisibly.

Method `start()`: Restarts a stopped Worker.

Usage:

```
Worker$start(wait = TRUE, timeout = NA)
```

Arguments:

wait If TRUE, blocks until the Worker is 'idle'. If FALSE, the Worker object is returned in the 'starting' state.

timeout How long to wait for the worker to finish starting (in seconds). If NA, defaults to the `Worker$new()` argument.

Returns: The Worker, invisibly.

Method `stop()`: Stops a Worker by terminating the background process and calling `<Job>$stop(reason)` on any Jobs currently assigned to this Worker.

Usage:

```
Worker$stop(reason = "worker stopped by user", cls = NULL)
```

Arguments:

reason Passed to `<Job>$stop()` for any Jobs currently managed by this Worker.

cls Passed to `<Job>$stop()` for any Jobs currently managed by this Worker.

Returns: The Worker, invisibly.

Method `restart()`: Restarts a Worker by calling `<Worker>$stop(reason)` and `<Worker>$start()` in succession.

Usage:

```
Worker$restart(
  wait = TRUE,
  timeout = NA,
  reason = "restarting worker",
  cls = NULL
)
```

Arguments:

wait If TRUE, blocks until the Worker is 'idle'. If FALSE, the Worker object is returned in the 'starting' state.

timeout How long to wait for the worker to finish starting (in seconds). If NA, defaults to the Worker\$new() argument.

reason Passed to <Job>\$stop() for any Jobs currently managed by this Worker.

cls Passed to <Job>\$stop() for any Jobs currently managed by this Worker.

Returns: The Worker, invisibly.

Method on(): Attach a callback function to execute when the Worker enters state.

Usage:

```
Worker$on(state, func)
```

Arguments:

state The name of a Worker state. Typically one of:

- '*' - Every time the state changes.
- '.next' - Only one time, the next time the state changes.
- 'starting' - Waiting for the background process to load.
- 'idle' - Waiting for Jobs to be \$run().
- 'busy' - While a Job is running.
- 'stopped' - After <Worker>\$stop() is called.

func A function that accepts a Worker object as input. You can call <Worker>\$stop() and other <Worker>\$ methods.

Returns: A function that when called removes this callback from the Worker.

Method wait(): Blocks until the Worker enters the given state.

Usage:

```
Worker$wait(state = "idle", timeout = Inf, signal = TRUE)
```

Arguments:

state The name of a Worker state. Typically one of:

- '*' - Every time the state changes.
- '.next' - Only one time, the next time the state changes.
- 'starting' - Waiting for the background process to load.
- 'idle' - Waiting for Jobs to be \$run().
- 'busy' - While a Job is running.
- 'stopped' - After <Worker>\$stop() is called.

timeout Stop the Worker if it takes longer than this number of seconds.

signal Raise an error if encountered (will also be recorded in <Worker>\$cnd).

Returns: This Worker, invisibly.

Method `run()`: Assigns a Job to this Worker for evaluation on the background process.

Usage:

```
Worker$run(job)
```

Arguments:

job A **Job** object, as created by `Job$new()`.

Returns: This Worker, invisibly.

Index

Job, [2](#), [7-9](#), [12](#)
Jobs, [5](#)

Queue, [3](#), [5](#)

Worker, [6](#), [9](#)
Workers, [5](#)