Package 'mirai'

August 16, 2022

Type Package

Title Minimalist Async Evaluation Framework for R

Version 0.5.3

Description Extremely simple and lightweight method for concurrent / parallel code execution, built on 'nanonext' and 'NNG' (Nanomsg Next Gen) technology.

License GPL (>= 3)

BugReports https://github.com/shikokuchuo/mirai/issues

URL https://shikokuchuo.net/mirai/, https://github.com/shikokuchuo/mirai/

Encoding UTF-8

Depends R (>= 2.4)

Imports nanonext (>= 0.5.3)

RoxygenNote 7.2.1

NeedsCompilation no

Author Charlie Gao [aut, cre] (<https://orcid.org/0000-0002-0750-061X>), Hibiki AI Limited [cph]

Maintainer Charlie Gao <charlie.gao@shikokuchuo.net>

Repository CRAN

Date/Publication 2022-08-16 21:30:02 UTC

R topics documented:

`mirai-package` *mirai: Minimalist Async Evaluation Framework for R*

Description

Extremely simple and lightweight method for concurrent / parallel code execution, built on 'nanonext' and 'NNG' (Nanomsg Next Gen) technology. mirai is Japanese for 'future'.

Links

mirai website: https://shikokuchuo.net/mirai/
mirai on CRAN: https://cran.r-project.org/package=mirai

nanonext website: https://shikokuchuo.net/nanonext/
nanonext on CRAN: https://cran.r-project.org/package=nanonext

NNG website: https://nng.nanomsg.org/

Author(s)

Charlie Gao <charlie.gao@shikokuchuo.net> (ORCID)

`call_mirai` *mirai (Call Value)*

Description

Call the value of a 'mirai', waiting for the the asynchronous operation to resolve if it is still in progress.

Usage

```
call_mirai(aio)
```

Arguments

`aio`	a 'mirai' (mirai are also aio objects).

Details

This function will wait for the async operation to complete if still in progress (blocking).

If an error occurs in evaluation, the error message is returned as a character string of class 'miraiError' and 'errorValue'. `is_mirai_error` may be used to test for this, otherwise `is_error_value` will also include other errors such as timeouts.

The 'mirai' updates itself in place, so to access the value of a 'mirai' `x` directly, use `call_mirai(x)$data`.

Value

The passed 'mirai' (invisibly). The retrieved value is stored at `$data`.

Alternatively

The value of a 'mirai' may be accessed at any time at `$data`, and if yet to resolve, an 'unresolved' logical NA will be returned instead.

`unresolved` may also be used on a 'mirai', and returns TRUE only if a 'mirai' has yet to resolve and FALSE otherwise. This is suitable for use in control flow statements such as `while` or `if`.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

m <- mirai(x + y + 1, x = 2, y = 3)
m
m$data
Sys.sleep(0.2)
m$data

df1 <- data.frame(a = 1, b = 2)
df2 <- data.frame(a = 3, b = 1)
m <- mirai(as.matrix(rbind(df1, df2)), .args = list(df1, df2), .timeout = 1000)
call_mirai(m)$data

m <- mirai({
  res <- rnorm(n)
  res / rev(res)
}, n = 1e6)
while (unresolved(m)) {
  cat("unresolved\n")
  Sys.sleep(0.1)
}
str(m$data)

}
```

daemons *daemons (Background Processes)*

Description

Set or view the number of daemons (background processes). Create persistent background processes to receive `mirai` requests. This provides a potentially more efficient solution for async operations as new processes no longer need to be created on an ad hoc basis.

Usage

```
daemons(...)
```

Arguments

`...`	either an integer to set the number of daemons, or 'view' to view the number of currently active daemons.

Details

{mirai} will revert to the default behaviour of creating a new background process for each request if the number of daemons is set to 0.

The current implementation is low-level and ensures tasks are evenly-distributed amongst daemons without actively managing a task queue. This approach provides a robust and resource-light solution, particularly well-suited to working with similar-length tasks, or where the number of concurrent tasks typically does not exceed the number of available daemons.

Value

Depending on the specified `...` parameter:

- integer: integer change in number of daemons (created or destroyed).
- 'view': integer number of currently set daemons.
- missing: the 'nanoSocket' for connecting to the daemons, or NULL if it is yet to be created.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

# Create 4 daemons
daemons(4)
# View the number of active daemons
daemons("view")
# Reset to zero
daemons(0)

}
```

`eval_mirai` *mirai (Evaluate Async)*

Description

Evaluate an expression asynchronously in a new background R process. This function will return immediately with a 'mirai', which will resolve to the evaluated result once complete.

Usage

```
eval_mirai(.expr, ..., .args = list(), .timeout = NULL)

mirai(.expr, ..., .args = list(), .timeout = NULL)
```

Arguments

`.expr`	an expression to evaluate in a new R process. This may be of arbitrary length, wrapped in { } if necessary.
`...`	(optional) named arguments specifying variables contained in '.expr'.
`.args`	(optional) list supplying arguments to '.expr' (used in addition to or instead of named arguments specified as '...').
`.timeout`	(optional) integer value in milliseconds or NULL for no timeout. A 'mirai' will resolve to an 'errorValue' 5 (timed out) if evaluation exceeds this limit.

Details

This function will return a 'mirai' object immediately.

The value of a 'mirai' may be accessed at any time at `$data`, and if yet to resolve, an 'unresolved' logical NA will be returned instead.

`unresolved` may also be used on a 'mirai', which returns TRUE only if a 'mirai' has yet to resolve and FALSE otherwise. This is suitable for use in control flow statements such as `while` or `if`.

Alternatively, to call (and wait for) the result, use `call_mirai` on the returned 'mirai' object. This will block until the result is returned.

The expression '.expr' will be evaluated in a new R process in a clean environment consisting of the named objects passed as '...' only (along with objects in the list '.args', if supplied).

If an error occurs in evaluation, the error message is returned as a character string of class 'miraiError' and 'errorValue'. `is_mirai_error` may be used to test for this, otherwise `is_error_value` will also include other errors such as timeouts.

`mirai` is an alias for `eval_mirai`.

Value

A 'mirai' object.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

m <- mirai(x + y + 1, x = 2, y = 3)
m
m$data
Sys.sleep(0.2)
m$data

df1 <- data.frame(a = 1, b = 2)
df2 <- data.frame(a = 3, b = 1)
m <- mirai(as.matrix(rbind(df1, df2)), .args = list(df1, df2), .timeout = 1000)
call_mirai(m)$data

m <- mirai({
  res <- rnorm(n)
  res / rev(res)
}, n = 1e6)
while (unresolved(m)) {
  cat("unresolved\n")
  Sys.sleep(0.1)
}
str(m$data)

}
```

---

is_error_value *Is Error Value*

Description

Is the object an error value generated by the system or a 'miraiError' from failed execution within a mirai. Includes user-specified errors such as mirai timeouts.

Usage

```
is_error_value(x)
```

Arguments

x	an object.

Value

Logical value TRUE if 'x' is of class 'errorValue', FALSE otherwise.

Examples

```
is_error_value(1L)
```

`is_mirai` *Is mirai*

Description

Is the object a mirai.

Usage

```
is_mirai(x)
```

Arguments

`x`	an object.

Value

Logical value TRUE or FALSE.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

m <- mirai(as.matrix(df), df = data.frame())
is_mirai(m)
is_mirai(df)

}
```

`is_mirai_error` *Is mirai Error*

Description

Is the object a 'miraiError'. When execution in a mirai process fails, the error message is returned as a character string of class 'miraiError' and 'errorValue'. To test for all errors, including timeouts etc., `is_error_value` should be used instead.

Usage

```
is_mirai_error(x)
```

Arguments

x	an object.

Value

Logical value TRUE if 'x' is of class 'miraiError', FALSE otherwise.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

m <- mirai(stop())
call_mirai(m)
is_mirai_error(m$data)

}
```

---

stop_mirai *mirai (Stop Evaluation)*

Description

Stop evaluation of a mirai that is in progress.

Usage

```
stop_mirai(aio)
```

Arguments

aio	a 'mirai' (mirai are also aio objects).

Details

Stops the asynchronous operation associated with 'mirai' by aborting, and then waits for it to complete or to be completely aborted. The 'mirai' is then deallocated and attempting to access the value at `$data` will result in an error.

Value

Invisible NULL.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

s <- mirai(Sys.sleep(n), n = 5)
stop_mirai(s)

}
```

`unresolved` *Query if a Mirai is Unresolved*

Description

Query whether a mirai or mirai value remains unresolved. Unlike `call_mirai`, this function does not wait for completion.

Usage

```
unresolved(aio)
```

Arguments

`aio`	A 'mirai' or mirai value stored in `$data` (mirai are also aio objects).

Details

Returns TRUE for unresolved mirai or mirai values, FALSE otherwise.

Suitable for use in control flow statements such as `while` or `if`.

Note: querying resolution may cause a previously unresolved mirai to resolve.

Value

Logical TRUE or FALSE.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

m <- mirai(Sys.sleep(0.1))
unresolved(m)
Sys.sleep(0.5)
unresolved(m)

}
```

`%>>%` *Deferred Evaluation Pipe*

Description

Pipe a possibly unresolved value forward into a function.

Usage

```
x %>>% f
```

Arguments

`x`	a value that is possibly an 'unresolvedValue'.
`f`	a function that accepts 'x' as its first argument.

Details

An 'unresolvedExpr' encapsulates the eventual evaluation result. Query its `$data` element for resolution. Once resolved, the object changes into a 'resolvedExpr' and the evaluated result will be available at `$data`.

Supports stringing together a series of piped expressions (as per the below example).

`unresolved` may be used on an 'unresolvedExpr' or its `$data` element to test for resolution.

Value

The evaluated result, or if x is an 'unresolvedValue', an 'unresolvedExpr'.

Usage

Usage is similar to R's native `|>` pipe.

`x %>>% f` is equivalent to `f(x)`

`x %>>% f()` is equivalent to `f(x)`

`x %>>% f(y)` is equivalent to `f(x, y)`

Please note that other usage is not supported and it is not a drop-in replacement for magrittr's `%>%` pipe.

Examples

```
if (interactive()) {
# Only run examples in interactive R sessions

m <- mirai({Sys.sleep(0.5); 1})
b <- m$data %>>% c(2, 3) %>>% as.character()
b
b$data
```

```
    Sys.sleep(1)
    b$data
    b

    }
```

Index