Package 'foreach'

October 13, 2022

Type Package

Title Provides Foreach Looping Construct

Version 1.5.2

Description Support for the foreach looping construct. Foreach is an idiom that allows for iterating over elements in a collection, without the use of an explicit loop counter. This package in particular is intended to be used for its return value, rather than for its side effects. In that sense, it is similar to the standard lapply function, but doesn't require the evaluation of a function. Using foreach without side effects also facilitates executing the loop in parallel.

License Apache License (== 2.0)

URL https://github.com/RevolutionAnalytics/foreach

BugReports https://github.com/RevolutionAnalytics/foreach/issues

Depends R (>= 2.5.0)

Imports codetools, utils, iterators

Suggests randomForest, doMC, doParallel, testthat, knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.1.1

Collate 'callCombine.R' 'foreach.R' 'do.R' 'foreach-ext.R' 'foreach-pkg.R' 'getDoPar.R' 'getDoSeq.R' 'getsyms.R' 'iter.R' 'nextElem.R' 'onLoad.R' 'setDoPar.R' 'setDoSeq.R' 'times.R' 'utils.R'

NeedsCompilation no

Author Folashade Daniel [cre],
Hong Ooi [ctb],
Rich Calaway [ctb],
Microsoft [aut, cph],
Steve Weston [aut]

Maintainer Folashade Daniel <fdaniel@microsoft.com>

Repository CRAN

Date/Publication 2022-02-02 09:20:02 UTC

R topics documented:

`foreach` *foreach*

Description

`%do%` and `%dopar%` are binary operators that operate on a `foreach` object and an `R` expression. The expression, `ex`, is evaluated multiple times in an environment that is created by the `foreach` object, and that environment is modified for each evaluation as specified by the `foreach` object. `%do%` evaluates the expression sequentially, while `%dopar%` evaluates it in parallel. The results of evaluating `ex` are returned as a list by default, but this can be modified by means of the `.combine` argument.

Usage

```
foreach(
  ...,
  .combine,
  .init,
  .final = NULL,
  .inorder = TRUE,
  .multicombine = FALSE,
  .maxcombine = if (.multicombine) 100 else 2,
  .errorhandling = c("stop", "remove", "pass"),
  .packages = NULL,
  .export = NULL,
  .noexport = NULL,
  .verbose = FALSE
)

e1 %:% e2

when(cond)

obj %do% ex

obj %dopar% ex
```

```
times(n)
```

Arguments

`...`	one or more arguments that control how `ex` is evaluated. Named arguments specify the name and values of variables to be defined in the evaluation environment. An unnamed argument can be used to specify the number of times that `ex` should be evaluated. At least one argument must be specified in order to define the number of times `ex` should be executed. If multiple arguments are supplied, the number of times `ex` is evaluated is equal to the smallest number of iterations among the supplied arguments. See the examples.
`.combine`	function that is used to process the tasks results as they generated. This can be specified as either a function or a non-empty character string naming the function. Specifying 'c' is useful for concatenating the results into a vector, for example. The values 'cbind' and 'rbind' can combine vectors into a matrix. The values '+' and '*' can be used to process numeric data. By default, the results are returned in a list.
`.init`	initial value to pass as the first argument of the `.combine` function. This should not be specified unless `.combine` is also specified.
`.final`	function of one argument that is called to return final result.
`.inorder`	logical flag indicating whether the `.combine` function requires the task results to be combined in the same order that they were submitted. If the order is not important, then it setting `.inorder` to `FALSE` can give improved performance. The default value is 'TRUE.
`.multicombine`	logical flag indicating whether the `.combine` function can accept more than two arguments. If an arbitrary `.combine` function is specified, by default, that function will always be called with two arguments. If it can take more than two arguments, then setting `.multicombine` to `TRUE` could improve the performance. The default value is `FALSE` unless the `.combine` function is `cbind`, `rbind`, or `c`, which are known to take more than two arguments.
`.maxcombine`	maximum number of arguments to pass to the combine function. This is only relevant if `.multicombine` is `TRUE`.
`.errorhandling`	specifies how a task evaluation error should be handled. If the value is "stop", then execution will be stopped via the `stop` function if an error occurs. If the value is "remove", the result for that task will not be returned, or passed to the `.combine` function. If it is "pass", then the error object generated by task evaluation will be included with the rest of the results. It is assumed that the combine function (if specified) will be able to deal with the error object. The default value is "stop".
`.packages`	character vector of packages that the tasks depend on. If `ex` requires a `R` package to be loaded, this option can be used to load that package on each of the workers. Ignored when used with `%do%`.
`.export`	character vector of variables to export. This can be useful when accessing a variable that isn't defined in the current environment. The default value in `NULL`.

.noexport	character vector of variables to exclude from exporting. This can be useful to prevent variables from being exported that aren't actually needed, perhaps because the symbol is used in a model formula. The default value in NULL.
.verbose	logical flag enabling verbose messages. This can be very useful for trouble shooting.
e1	foreach object to merge.
e2	foreach object to merge.
cond	condition to evaluate.
obj	foreach object used to control the evaluation of ex.
ex	the R expression to evaluate.
n	number of times to evaluate the R expression.

Details

The foreach and %do%/%dopar% operators provide a looping construct that can be viewed as a hybrid of the standard for loop and lapply function. It looks similar to the for loop, and it evaluates an expression, rather than a function (as in lapply), but its purpose is to return a value (a list, by default), rather than to cause side-effects. This facilitates parallelization, but looks more natural to people that prefer for loops to lapply.

The %:% operator is the *nesting* operator, used for creating nested foreach loops. Type vignette("nested") at the R prompt for more details.

Parallel computation depends upon a *parallel backend* that must be registered before performing the computation. The parallel backends available will be system-specific, but include doParallel, which uses R's built-in **parallel** package. Each parallel backend has a specific registration function, such as registerDoParallel.

The times function is a simple convenience function that calls foreach. It is useful for evaluating an R expression multiple times when there are no varying arguments. This can be convenient for resampling, for example.

See Also

iterators::iter

Examples

```
# equivalent to rnorm(3)
times(3) %do% rnorm(1)

# equivalent to lapply(1:3, sqrt)
foreach(i=1:3) %do%
  sqrt(i)

# multiple ... arguments
foreach(i=1:4, j=1:10) %do%
sqrt(i+j)

# equivalent to colMeans(m)
```

```
m <- matrix(rnorm(9), 3, 3)
foreach(i=1:ncol(m), .combine=c) %do%
  mean(m[,i])

# normalize the rows of a matrix in parallel, with parenthesis used to
# force proper operator precedence
# Need to register a parallel backend before this example will run
# in parallel
foreach(i=1:nrow(m), .combine=rbind) %dopar%
  (m[i,] / mean(m[i,]))

# simple (and inefficient) parallel matrix multiply
library(iterators)
a <- matrix(1:16, 4, 4)
b <- t(a)
foreach(b=iter(b, by='col'), .combine=cbind) %dopar%
  (a %*% b)

# split a data frame by row, and put them back together again without
# changing anything
d <- data.frame(x=1:10, y=rnorm(10))
s <- foreach(d=iter(d, by='row'), .combine=rbind) %dopar% d
identical(s, d)

# a quick sort function
qsort <- function(x) {
  n <- length(x)
  if (n == 0) {
    x
  } else {
    p <- sample(n, 1)
    smaller <- foreach(y=x[-p], .combine=c) %:% when(y <= x[p]) %do% y
    larger  <- foreach(y=x[-p], .combine=c) %:% when(y >  x[p]) %do% y
    c(qsort(smaller), x[p], qsort(larger))
  }
}
qsort(runif(12))
```

`foreach-ext` *foreach extension functions*

Description

These functions are used to write parallel backends for the `foreach` package. They should not be used from normal scripts or packages that use the `foreach` package.

Usage

```
makeAccum(it)
```

```
accumulate(obj, result, tag, ...)

getResult(obj, ...)

getErrorValue(obj, ...)

getErrorIndex(obj, ...)

## S3 method for class 'iforeach'
accumulate(obj, result, tag, ...)

## S3 method for class 'iforeach'
getResult(obj, ...)

## S3 method for class 'iforeach'
getErrorValue(obj, ...)

## S3 method for class 'iforeach'
getErrorIndex(obj, ...)

## S3 method for class 'ixforeach'
accumulate(obj, result, tag, ...)

## S3 method for class 'ixforeach'
getResult(obj, ...)

## S3 method for class 'ixforeach'
getErrorValue(obj, ...)

## S3 method for class 'ixforeach'
getErrorIndex(obj, ...)

## S3 method for class 'ifilteredforeach'
accumulate(obj, result, tag, ...)

## S3 method for class 'ifilteredforeach'
getResult(obj, ...)

## S3 method for class 'ifilteredforeach'
getErrorValue(obj, ...)

## S3 method for class 'ifilteredforeach'
getErrorIndex(obj, ...)

getexports(ex, e, env, good = character(0), bad = character(0))
```

Arguments

it	foreach iterator.
obj	foreach iterator object.
result	task result to accumulate.
tag	tag of task result to accumulate.
...	unused.
ex	call object to analyze.
e	local environment of the call object.
env	exported environment in which call object will be evaluated.
good	names of symbols that are being exported.
bad	names of symbols that are not being exported.

Note

These functions are likely to change in future versions of the foreach package. When they become more stable, they will be documented.

---

getDoParWorkers *Functions Providing Information on the doPar Backend*

Description

The getDoParWorkers function returns the number of execution workers there are in the currently registered doPar backend. It can be useful when determining how to split up the work to be executed in parallel. A 1 is returned by default.

The getDoParRegistered function returns TRUE if a doPar backend has been registered, otherwise FALSE.

The getDoParName function returns the name of the currently registered doPar backend. A NULL is returned if no backend is registered.

The getDoParVersion function returns the version of the currently registered doPar backend. A NULL is returned if no backend is registered.

Usage

```
getDoParWorkers()

getDoParRegistered()

getDoParName()

getDoParVersion()
```

Examples

```
cat(sprintf('%s backend is registered\n',
            if(getDoParRegistered()) 'A' else 'No'))
cat(sprintf('Running with %d worker(s)\n', getDoParWorkers()))
(name <- getDoParName())
(ver <- getDoParVersion())
if (getDoParRegistered())
  cat(sprintf('Currently using %s [%s]\n', name, ver))
```

`getDoSeqWorkers` *Functions Providing Information on the doSeq Backend*

Description

The `getDoSeqWorkers` function returns the number of execution workers there are in the currently registered doSeq backend. A 1 is returned by default.

The `getDoSeqRegistered` function returns TRUE if a doSeq backend has been registered, otherwise FALSE.

The `getDoSeqName` function returns the name of the currently registered doSeq backend. A `NULL` is returned if no backend is registered.

The `getDoSeqVersion` function returns the version of the currently registered doSeq backend. A `NULL` is returned if no backend is registered.

Usage

```
getDoSeqRegistered()

getDoSeqWorkers()

getDoSeqName()

getDoSeqVersion()
```

Examples

```
cat(sprintf('%s backend is registered\n',
            if(getDoSeqRegistered()) 'A' else 'No'))
cat(sprintf('Running with %d worker(s)\n', getDoSeqWorkers()))
(name <- getDoSeqName())
(ver <- getDoSeqVersion())
if (getDoSeqRegistered())
  cat(sprintf('Currently using %s [%s]\n', name, ver))
```

registerDoSEQ *registerDoSEQ*

Description

The registerDoSEQ function is used to explicitly register a sequential parallel backend with the foreach package. This will prevent a warning message from being issued if the %dopar% function is called and no parallel backend has been registered.

Usage

```
registerDoSEQ()
```

See Also

doParallel::registerDoParallel

Examples

```
# specify that %dopar% should run sequentially
registerDoSEQ()
```

setDoPar *setDoPar*

Description

The setDoPar function is used to register a parallel backend with the foreach package. This isn't normally executed by the user. Instead, packages that provide a parallel backend provide a function named registerDoPar that calls setDoPar using the appropriate arguments.

Usage

```
setDoPar(fun, data = NULL, info = function(data, item) NULL)
```

Arguments

fun	A function that implements the functionality of %dopar%.
data	Data to be passed to the registered function.
info	Function that retrieves information about the backend.

See Also

%dopar%

setDoSeq *setDoSeq*

Description

The setDoSeq function is used to register a sequential backend with the foreach package. This isn't normally executed by the user. Instead, packages that provide a sequential backend provide a function named registerDoSeq that calls setDoSeq using the appropriate arguments.

Usage

```
setDoSeq(fun, data = NULL, info = function(data, item) NULL)
```

Arguments

fun	A function that implements the functionality of %dopar%.
data	Data to be passed to the registered function.
info	Function that retrieves information about the backend.

See Also

%dopar%

Index