Package 'iterators'

October 15, 2020

Type Package

Title Provides Iterator Construct

Version 1.0.13

Description Support for iterators, which allow a programmer to traverse through all the elements of a vector, list, or other collection of data.

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

Depends R (>= 2.5.0), utils

Suggests RUnit, foreach

License Apache License (== 2.0)

NeedsCompilation no

Author Michelle Wallig [cre],
Revolution Analytics [aut, cph],
Steve Weston [aut]

Maintainer Michelle Wallig <Michelle.Wallig@microsoft.com>

Repository CRAN

Date/Publication 2020-10-15 05:00:03 UTC

R topics documented:

iterators-package *The Iterators Package*

Description

The iterators package provides tools for iterating over various R data structures. Iterators are available for vectors, lists, matrices, data frames, and files. By following very simple conventions, new iterators can be written to support any type of data source, such as database queries or dynamically generated data.

Details

Further information is available in the following help topics:

`iter`	Generic function used to create iterator objects.
`nextElem`	Generic function used to get the next element of a iterator.
`icount`	A function used to create a counting iterator.
`idiv`	A function used to create a number dividing iterator.
`ireadLines`	A function used to create a file reading iterator.

For a complete list of functions with individual help pages, use `library(help="iterators")`.

iapply *Array/Apply Iterator*

Description

Returns an iterator over an array, which iterates over the array in much the same manner as the `apply` function.

Usage

```
iapply(X, MARGIN)
```

Arguments

`X`	the array to iterate over.
`MARGIN`	a vector of subscripts. `1` indicates the first dimension (rows), `2` indicates the second dimension (columns), etc.

Value

The apply iterator.

See Also

apply

Examples

```
a <- array(1:8, c(2, 2, 2))

# iterate over all the matrices
it <- iapply(a, 3)
as.list(it)

# iterate over all the columns of all the matrices
it <- iapply(a, c(2, 3))
as.list(it)

# iterate over all the rows of all the matrices
it <- iapply(a, c(1, 3))
as.list(it)
```

---

icount *Counting Iterators*

Description

Returns an iterator that counts starting from one.

Usage

```
icount(count)
icountn(vn)
```

Arguments

count	number of times that the iterator will fire. If not specified, it will count forever.
vn	vector of counts.

Value

The counting iterator.

Examples

```
# create an iterator that counts from 1 to 3.
it <- icount(3)
nextElem(it)
nextElem(it)
nextElem(it)
try(nextElem(it))  # expect a StopIteration exception
```

idiv *Dividing Iterator*

Description

Returns an iterator that returns pieces of numeric value.

Usage

```
idiv(n, ..., chunks, chunkSize)
```

Arguments

n	number of times that the iterator will fire. If not specified, it will count forever.
...	unused.
chunks	the number of pieces that n should be divided into. This is useful when you know the number of pieces that you want. If specified, then chunkSize should not be.
chunkSize	the maximum size of the pieces that n should be divided into. This is useful when you know the size of the pieces that you want. If specified, then chunks should not be.

Value

The dividing iterator.

Examples

```
# divide the value 10 into 3 pieces
it <- idiv(10, chunks=3)
nextElem(it)
nextElem(it)
nextElem(it)
try(nextElem(it))  # expect a StopIteration exception

# divide the value 10 into pieces no larger than 3
it <- idiv(10, chunkSize=3)
nextElem(it)
nextElem(it)
nextElem(it)
nextElem(it)
try(nextElem(it))  # expect a StopIteration exception
```

iread.table *Iterator over Rows of a Data Frame Stored in a File*

Description

Returns an iterator over the rows of a data frame stored in a file in table format. It is a wrapper around the standard `read.table` function.

Usage

```
iread.table(file, ..., verbose=FALSE)
```

Arguments

`file`	the name of the file to read the data from.
`...`	all additional arguments are passed on to the `read.table` function. See the documentation for `read.table` for more information.
`verbose`	logical value indicating whether or not to print the calls to `read.table`.

Value

The file reading iterator.

Note

In this version of `iread.table`, both the `read.table` arguments `header` and `row.names` must be specified. This is because the default values of these arguments depend on the contents of the beginning of the file. In order to make the subsequent calls to `read.table` work consistently, the user must specify those arguments explicitly. A future version of `iread.table` may remove this requirement.

See Also

`read.table`

ireadLines *Iterator over Lines of Text from a Connection*

Description

Returns an iterator over the lines of text from a connection. It is a wrapper around the standard `readLines` function.

Usage

```
ireadLines(con, n=1, ...)
```

Arguments

con	a connection object or a character string.
n	integer. The maximum number of lines to read. Negative values indicate that one should read up to the end of the connection. The default value is 1.
...	passed on to the readLines function.

Value

The line reading iterator.

See Also

readLines

Examples

```
# create an iterator over the lines of COPYING
it <- ireadLines(file.path(R.home(), 'COPYING'))
nextElem(it)
nextElem(it)
nextElem(it)
```

---

irnorm *Random Number Iterators*

Description

These function returns an iterators that return random numbers of various distributions. Each one is a wrapper around a standard R function.

Usage

```
irnorm(..., count)
irunif(..., count)
irbinom(..., count)
irnbinom(..., count)
irpois(..., count)
```

Arguments

count	number of times that the iterator will fire. If not specified, it will fire values forever.
...	arguments to pass to the underlying rnorm function.

Value

An iterator that is a wrapper around the corresponding random number generator function.

Examples

```
# create an iterator that returns three random numbers
it <- irnorm(1, count=3)
nextElem(it)
nextElem(it)
nextElem(it)
try(nextElem(it))  # expect a StopIteration exception
```

`isplit` *Split Iterator*

Description

Returns an iterator that divides the data in the vector `x` into the groups defined by `f`.

Usage

```
isplit(x, f, drop=FALSE, ...)
```

Arguments

`x`	vector or data frame of values to be split into groups.
`f`	a factor or list of factors used to categorize `x`.
`drop`	logical indicating if levels that do not occur should be dropped.
`...`	current ignored.

Value

The split iterator.

See Also

`split`

Examples

```
x <- rnorm(200)
f <- factor(sample(1:10, length(x), replace=TRUE))

it <- isplit(x, f)
expected <- split(x, f)

for (i in expected) {
  actual <- nextElem(it)
  stopifnot(actual$value == i)
}
```

iter *Iterator Factory Functions*

Description

iter is a generic function used to create iterator objects.

Usage

```
iter(obj, ...)

## Default S3 method:
iter(obj, checkFunc=function(...) TRUE, recycle=FALSE,
...)
## S3 method for class 'iter'
iter(obj, ...)
## S3 method for class 'matrix'
iter(obj, by=c('column', 'cell', 'row'), chunksize=1L,
checkFunc=function(...) TRUE, recycle=FALSE, ...)
## S3 method for class 'data.frame'
iter(obj, by=c('column', 'row'),
checkFunc=function(...) TRUE, recycle=FALSE, ...)
## S3 method for class 'function'
iter(obj, checkFunc=function(...) TRUE,
recycle=FALSE, ...)
```

Arguments

obj	an object from which to generate an iterator.
by	how to iterate.
chunksize	the number of elements of by to return with each call to nextElem.
checkFunc	a function which, when passed an iterator value, return TRUE or FALSE. If FALSE, the value is skipped in the iteration.
recycle	a boolean describing whether the iterator should reset after running through all it's values.
...	additional arguments affecting the iterator.

Value

The iterator.

Examples

```
# a vector iterator
i1 <- iter(1:3)
nextElem(i1)
```

```
nextElem(i1)
nextElem(i1)

# a vector iterator with a checkFunc
i1 <- iter(1:3, checkFunc=function(i) i %% 2 == 0)
nextElem(i1)

# a data frame iterator by column
i2 <- iter(data.frame(x=1:3, y=10, z=c('a', 'b', 'c')))
nextElem(i2)
nextElem(i2)
nextElem(i2)

# a data frame iterator by row
i3 <- iter(data.frame(x=1:3, y=10), by='row')
nextElem(i3)
nextElem(i3)
nextElem(i3)

# a function iterator
i4 <- iter(function() rnorm(1))
nextElem(i4)
nextElem(i4)
nextElem(i4)
```

`makeIwrapper` *Iterator Maker Generator*

Description

The `makeIwrapper` function makes iterator makers. The resulting iterator makers all take an optional `count` argument which specifies the number of times the resulting iterator should fire. The iterators are wrappers around functions that return different values each time they are called. The `isample` function is an example of one such iterator maker (as are `irnorm`, `irunif`, etc.).

Usage

```
makeIwrapper(FUN)
isample(..., count)
```

Arguments

`FUN`	a character string naming a function that generates different values each time it is called; typically one of the standard random number generator functions.
`count`	number of times that the iterator will fire. If not specified, it will fire values forever.
`...`	arguments to pass to the underlying `FUN` function.

Value

An iterator that is a wrapper around the corresponding function.

Examples

```
# create an iterator maker for the sample function
mysample <- makeIwrapper('sample')
# use this iterator maker to generate an iterator
# that will generate three five member samples from the
# sequence 1:100
it <- mysample(1:100, 5, count=3)
nextElem(it)
nextElem(it)
nextElem(it)
try(nextElem(it))  # expect a StopIteration exception
```

`nextElem` *Get Next Element of Iterator*

Description

`nextElem` is a generic function used to produce values. If a `checkFunc` was specified to the constructor, the potential iterated values will be passed to the `checkFunc` until the `checkFunc` returns `TRUE`. When the iterator has no more values, it calls stop with the message 'StopIteration'.

Usage

```
nextElem(obj, ...)

## S3 method for class 'containeriter'
nextElem(obj, ...)
## S3 method for class 'funiter'
nextElem(obj, ...)
```

Arguments

`obj`	an iterator object.
`...`	additional arguments that are ignored.

Value

The value.

Examples

```
it <- iter(c('a', 'b', 'c'))
print(nextElem(it))
print(nextElem(it))
print(nextElem(it))
```

Index