

Package ‘itertools’

February 14, 2012

Type Package

Title Iterator Tools

Version 0.1-1

Author Steve Weston, Hadley Wickham

Maintainer Steve Weston <stephen.b.weston@gmail.com>

Description Various tools for creating iterators, many patterned after functions in the Python itertools module, and others patterned after functions in the ‘snow’ package.

Depends R (>= 2.5.0), iterators(>= 1.0.0)

Suggests foreach

Enhances foreach, iterators

License GPL-2

Repository CRAN

Repository/R-Forge/Project itertools

Repository/R-Forge/Revision 38

Date/Publication 2010-01-14 15:37:23

R topics documented:

itertools-package	2
chain	3
enumerate	3
hasNext	4
ichunk	4
ifilter	5
ihasNext	6
ilimit	6

ireadBin	7
irep	8
irepeat	8
is.iterator	9
isplitCols	10
isplitIndices	11
isplitRows	12
isplitVector	13
izip	14
product	14
recycle	15
timeout	15

Index	16
--------------	-----------

itertools-package	<i>The itertools Package</i>
-------------------	------------------------------

Description

The `itertools` package provides a variety of functions used to create iterators, as defined by REvolution Computing’s `iterators` package. Many of the functions are patterned after functions of the same name in the Python `itertools` module, including `chain`, `product`, `izip`, `ifilter`, etc. In addition, a number of functions were inspired by utility functions in the `snow` package, such as `isplitRows`, `isplitCols`, and `isplitIndices`.

There are also several utility functions that were contributed by Hadley Wickham that aid in writing iterators. These include `is.iterator`, `end_iterator`, `iteration_has_ended`, and `new_iterator`.

Details

More information is available on the following topics:

<code>isplitVector</code>	splits, or slices, a vector into shorter segments
<code>isplitCols</code>	splits a matrix column-wise
<code>isplitRows</code>	splits a matrix row-wise
<code>isplitIndices</code>	iterate over “chunks” of indices from 1 to n
<code>chain</code>	chain multiple iterators together into one iterator
<code>enumerate</code>	create an enumeration from an iterator
<code>ichunk</code>	create lists of values from an iterator to aid manual chunking
<code>ihasNext</code>	add a <code>hasNext</code> method to an iterator
<code>ifilter</code>	only return values for which a predicate function returns true
<code>ifilterfalse</code>	only return values for which a predicate function returns false
<code>ilimit</code>	limit, or truncate, an iterator
<code>ireadBin</code>	reads from a binary connection
<code>irep</code>	an iterator version of the <code>rep</code> function
<code>irepeat</code>	a simple repeating value iterator
<code>izip</code>	zip together multiple iterators
<code>product</code>	zip together multiple iterators in cartesian product fashion

recycle	recycle values from an iterator repeatedly
timeout	iterate for a specified number of seconds
is_iterator	indicates if an object is an iterator
end_iteration	throws an exception to signal end of iteration
iteration_has_ended	tests an exception to see if iteration has ended
new_iterator	creates a new iterator object

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

chain *Create a chaining iterator*

Description

Create an iterator that chains multiple iterables together.

Usage

```
chain(...)
```

Arguments

... The iterables to iterate over.

Examples

```
# Iterate over two iterables
as.list(chain(1:2, letters[1:3]))
```

enumerate *Create an enumeration object*

Description

Create an iterator that iterates over an iterable, returning the value in a list that includes an index.

Usage

```
enumerate(iterable)
```

Arguments

iterable Iterable to iterate over.

Examples

```
# Create an enumeration of five random numbers
as.list(enumerate(rnorm(5)))
```

hasNext	<i>Does This Iterator Have A Next Element</i>
---------	---

Description

hasNext is a generic function that indicates if the iterator has another element.

Usage

```
hasNext(obj, ...)
```

```
## S3 method for class 'ihasNext'
hasNext(obj, ...)
```

Arguments

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

Value

Logical value indicating whether the iterator has a next element.

Examples

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

ichunk	<i>Create a chunking iterator</i>
--------	-----------------------------------

Description

Create an iterator that issues lists of values from the underlying iterable. This is useful for manually “chunking” values from an iterable.

Usage

```
ichunk(iterable, chunkSize)
```

Arguments

iterable	Iterable to iterate over.
chunkSize	Maximum number of values from iterable to return in each value issued by the resulting iterator.

See Also[isplitVector](#)**Examples**

```
# Split the vector 1:10 into "chunks" with a maximum length of three
it <- ihasNext(ichunk(1:10, 3))
while (hasNext(it)) {
  print(unlist(nextElem(it)))
}
```

ifilter*Create a filtering iterator*

Description

The `ifilter` and `ifilterfalse` functions create iterators that return a subset of the values of the specified iterable. `ifilter` returns the values for which the `pred` function returns `TRUE`, and `ifilterfalse` returns the values for which the `pred` function returns `FALSE`.

Usage

```
ifilter(pred, iterable)
ifilterfalse(pred, iterable)
```

Arguments

<code>pred</code>	A function that takes one argument and returns <code>TRUE</code> or <code>FALSE</code> .
<code>iterable</code>	The iterable to iterate over.

Examples

```
# Return the odd numbers between 1 and 10
as.list(ifilter(function(x) x %% 2 == 1, icount(10)))

# Return the even numbers between 1 and 10
as.list(ifilterfalse(function(x) x %% 2 == 1, icount(10)))
```

`ihasNext`*Create an iterator that supports the hasNext method*

Description

`ihasNext` is a generic function that indicates if the iterator has another element.

Usage

```
ihasNext(iterable)
```

Arguments

`iterable` an iterable object, which could be an iterator.

Value

An `ihasNext` iterator that wraps the specified iterator and supports the `hasNext` method.

Examples

```
it <- ihasNext(c('a', 'b', 'c'))
while (hasNext(it))
  print(nextElem(it))
```

`ilimit`*Create a limited iterator*

Description

Create an iterator that wraps a specified iterable a limited number of times.

Usage

```
ilimit(iterable, n)
```

Arguments

`iterable` Iterable to iterate over.
`n` Maximum number of values to return.

Examples

```
# Limit icount to only return three values
as.list(ilimit(icontains(), 3))
```

ireadBin *Create an iterator to read binary data from a connection*

Description

Create an iterator to read binary data from a connection.

Usage

```
ireadBin(con, what='raw', n=1L, size=NA_integer_, signed=TRUE,
         endian=.Platform$endian, ipos=NULL)
```

Arguments

con	A connection object or a character string naming a file or a raw vector.
what	Either an object whose mode will give the mode of the vector to be read, or a character vector of length one describing the mode: one of “numeric”, “double”, “integer”, “int”, “logical”, “complex”, “character”, “raw”. Unlike readBin, the default value is “raw”.
n	integer. The (maximal) number of records to be read each time the iterator is called.
size	integer. The number of bytes per element in the byte stream. The default, ‘NA_integer_’, uses the natural size.
signed	logical. Only used for integers of sizes 1 and 2, when it determines if the quantity on file should be regarded as a signed or unsigned integer.
endian	The endian-ness (“big” or “little”) of the target system for the file. Using “swap” will force swapping endian-ness.
ipos	iterable. If not NULL, values from this iterable will be used to do a seek on the file before calling readBin.

Examples

```
zz <- file("testbin", "wb")
writeBin(1:100, zz)
close(zz)

it <- ihasNext(ireadBin("testbin", integer(), 10))
while (hasNext(it)) {
  print(nextElem(it))
}
unlink("testbin")
```

irep *Create a repeating iterator*

Description

Create an iterator version of the rep function.

Usage

```
irep(iterable, times, length.out, each)
```

Arguments

iterable	The iterable to iterate over repeatedly.
times	A vector giving the number of times to repeat each element if the length is greater than one, or to repeat all the elements if the length is one. This behavior is less strict than rep since the length of an iterable isn't generally known.
length.out	non-negative integer. The desired length of the output iterator.
each	non-negative integer. Each element of the iterable is repeated each times.

See Also

[rep](#)

Examples

```
unlist(as.list(irep(1:4, 2)))
unlist(as.list(irep(1:4, each=2)))
unlist(as.list(irep(1:4, c(2,2,2,2))))
unlist(as.list(irep(1:4, c(2,1,2,1))))
unlist(as.list(irep(1:4, each=2, len=4)))
unlist(as.list(irep(1:4, each=2, len=10)))
unlist(as.list(irep(1:4, each=2, times=3)))
```

irepeat *Create a repeating iterator*

Description

Create an iterator that returns a value a specified number of times.

Usage

```
irepeat(x, times)
```

Arguments

x	The value to return repeatedly.
times	The number of times to repeat the value. Default value is infinity.

Examples

```
# Repeat a value 10 times
unlist(as.list(irepeat(42, 10)))
```

is.iterator

Utilities for writing iterators

Description

is.iterator indicates if an object is an iterator. end_iteration throws an exception to signal that there are no more values available in an iterator. iteration_has_ended tests an exception to see if it indicates that iteration has ended. new_iterator returns an iterator object.

Usage

```
is.iterator(x)
end_iteration()
iteration_has_ended(e)
new_iterator(nextElem, ...)
```

Arguments

x	any object.
e	a condition object.
nextElem	a function object that takes no arguments.
...	not currently used.

Examples

```
# Manually iterate using the iteration_has_ended function to help
it <- iter(1:3)
tryCatch({
  stopifnot(is.iterator(it))
  repeat {
    print(nextElem(it))
  }
},
error=function(e) {
  if (!iteration_has_ended(e)) {
    stop(e)
  }
})
```

`isplitCols`*Create an iterator that splits a matrix into block columns*

Description

Create an iterator that splits a matrix into block columns. You can specify either the number of blocks, using the `chunks` argument, or the maximum size of the blocks, using the `chunkSize` argument.

Usage

```
isplitCols(x, ...)
```

Arguments

<code>x</code>	Matrix to iterate over.
<code>...</code>	Passed as the second and subsequent arguments to <code>idiv</code> function. Currently, <code>idiv</code> accepts either a value for <code>chunks</code> or <code>chunkSize</code> .

Value

An iterator that returns submatrices of `x`.

See Also

[idiv](#), [isplitRows](#)

Examples

```
# Split a matrix into submatrices with a maximum of three columns
x <- matrix(1:30, 3)
it <- ihasNext(isplitCols(x, chunkSize=3))
while (hasNext(it)) {
  print(nextElem(it))
}

# Split the same matrix into five submatrices
it <- ihasNext(isplitCols(x, chunks=5))
while (hasNext(it)) {
  print(nextElem(it))
}
```

isplitIndices	<i>Create an iterator of indices</i>
---------------	--------------------------------------

Description

Create an iterator of chunks of indices from 1 to n. You can specify either the number of pieces, using the chunks argument, or the maximum size of the pieces, using the chunkSize argument.

Usage

```
isplitIndices(n, ...)
```

Arguments

n	Maximum index to generate.
...	Passed as the second and subsequent arguments to idiv function. Currently, idiv accepts either a value for chunks or chunkSize.

Value

An iterator that returns vectors of indices from 1 to n.

See Also

[idiv](#), [isplitVector](#)

Examples

```
# Return indices from 1 to 17 in vectors no longer than five
it <- ihasNext(isplitIndices(17, chunkSize=5))
while (hasNext(it)) {
  print(nextElem(it))
}

# Return indices from 1 to 7 in four vectors
it <- ihasNext(isplitIndices(7, chunks=4))
while (hasNext(it)) {
  print(nextElem(it))
}
```

`isplitRows`*Create an iterator that splits a matrix into block rows*

Description

Create an iterator that splits a matrix into block rows. You can specify either the number of blocks, using the `chunks` argument, or the maximum size of the blocks, using the `chunkSize` argument.

Usage

```
isplitRows(x, ...)
```

Arguments

<code>x</code>	Matrix to iterate over.
<code>...</code>	Passed as the second and subsequent arguments to <code>idiv</code> function. Currently, <code>idiv</code> accepts either a value for <code>chunks</code> or <code>chunkSize</code> .

Value

An iterator that returns submatrices of `x`.

See Also

[idiv](#), [isplitCols](#)

Examples

```
# Split a matrix into submatrices with a maximum of three rows
x <- matrix(1:100, 10)
it <- ihasNext(isplitRows(x, chunkSize=3))
while (hasNext(it)) {
  print(nextElem(it))
}

# Split the same matrix into five submatrices
it <- ihasNext(isplitRows(x, chunks=5))
while (hasNext(it)) {
  print(nextElem(it))
}
```

`isplitVector`*Create an iterator that splits a vector*

Description

Create an iterator that splits a vector into smaller pieces. You can specify either the number of pieces, using the `chunks` argument, or the maximum size of the pieces, using the `chunkSize` argument.

Usage

```
isplitVector(x, ...)
```

Arguments

<code>x</code>	Vector to iterate over. Note that it doesn't need to be an atomic vector, so a list is acceptable.
<code>...</code>	Passed as the second and subsequent arguments to <code>idiv</code> function. Currently, <code>idiv</code> accepts either a value for <code>chunks</code> or <code>chunkSize</code> .

Value

An iterator that returns vectors of the same type as `x` with one or more elements from `x`.

See Also

[idiv](#)

Examples

```
# Split the vector 1:10 into "chunks" with a maximum length of three
it <- ihasNext(isplitVector(1:10, chunkSize=3))
while (hasNext(it)) {
  print(nextElem(it))
}

# Split the vector "letters" into four chunks
it <- ihasNext(isplitVector(letters, chunks=4))
while (hasNext(it)) {
  print(nextElem(it))
}

# Get the first five elements of a list as a list
nextElem(isplitVector(as.list(letters), chunkSize=5))
```

 izip

Create an iterator over multiple iterables

Description

Create an iterator that iterates over multiple iterables, returning the values as a list.

Usage

```
izip(...)
```

Arguments

... The iterables to iterate over.

Examples

```
# Iterate over two iterables of different sizes
as.list(izip(a=1:2, b=letters[1:3]))
```

 product

Create a cartesian product iterator

Description

Create an iterator that returns values from multiple iterators in cartesian product fashion. That is, they are combined the manner of nested for loops.

Usage

```
product(...)
```

Arguments

... Named iterables to iterate over. The right-most iterables change more quickly, like an odometer.

Examples

```
# Simulate a doubly-nested loop with a single while loop
it <- ihasNext(product(a=1:3, b=1:2))
while (hasNext(it)) {
  x <- nextElem(it)
  cat(sprintf('a = %d, b = %d\n', x$a, x$b))
}
```

recycle	<i>Create a recycling iterator</i>
---------	------------------------------------

Description

Create an iterator that recycles a specified iterable.

Usage

```
recycle(iterable, times=NA_integer_)
```

Arguments

iterable	The iterable to recycle.
times	integer. Number of times to recycle the values in the iterator. Default value of NA_integer_ means to recycle forever.

Examples

```
# Recycle over 'a', 'b', and 'c' three times
recycle(letters[1:3], 3)
```

timeout	<i>Create a timeout iterator</i>
---------	----------------------------------

Description

Create an iterator that iterates over another iterator for a specified period of time, and then stops. This can be useful when you want to search for something, or run a test for awhile, and then stop.

Usage

```
timeout(iterable, time)
```

Arguments

iterable	Iterable to iterate over.
time	The time interval to iterate for, in seconds.

Examples

```
# See how high we can count in a tenth of a second
length(as.list(timeout(icount(), 0.1)))
```

Index

*Topic **methods**

hasNext, 4

*Topic **package**

itertools-package, 2

*Topic **utilities**

chain, 3

enumerate, 3

ichunk, 4

ifilter, 5

ihhasNext, 6

ilimit, 6

ireadBin, 7

irep, 8

irepeat, 8

is.iterator, 9

isplitCols, 10

isplitIndices, 11

isplitRows, 12

isplitVector, 13

izip, 14

product, 14

recycle, 15

timeout, 15

chain, 3

end_iteration(is.iterator), 9

enumerate, 3

hasNext, 4

ichunk, 4

idiv, 10–13

ifilter, 5

ifilterfalse(ifilter), 5

ihhasNext, 6

ilimit, 6

ireadBin, 7

irep, 8

irepeat, 8

is.iterator, 9

isplitCols, 10, 12

isplitIndices, 11

isplitRows, 10, 12

isplitVector, 5, 11, 13

iteration_has_ended(is.iterator), 9

itertools(itertools-package), 2

itertools-package, 2

izip, 14

new_iterator(is.iterator), 9

product, 14

recycle, 15

rep, 8

timeout, 15