Package ‘container’

December 1, 2018

Type Package
Title Deque, Set, and Dict - R6 Based Container Classes with Iterators
Version 0.3.0
Date 2018-12-01
Description Common container data structures deque, set and dict (resembling 'Python's dict type) with typical member functions to insert, delete and access container elements. Provides iterators and reference semantics.
Depends R (>= 3.0)
License GPL-3
LazyData TRUE
Encoding UTF-8
Imports R6
VignetteBuilder knitr
Suggests knitr, testthat, rmarkdown
URL https://github.com/rpahl/container
BugReports https://github.com/rpahl/container/issues
NeedsCompilation no
Author Roman Pahl [aut, cre]
Maintainer Roman Pahl <roman.pahl@gmail.com>
Repository CRAN
RoxygenNote 6.0.1
Date/Publication 2018-12-01 22:50:05 UTC

R topics documented:

Container .................................................. 2
container.pkg .......................................... 4
ContainerS3 ............................................. 4
ContainerS3op .......................................... 6
Container

A sequence container

Description

This class implements a container data structure with typical member functions to insert, delete and access objects from the container. It also serves as the base class for Deque, Set, and Dict.

Usage

Container

Format

An object of class R6ClassGenerator of length 24.

Details

The underlying data structure is based on R vectors (or lists), with the mode being set to the mode (or type) of the value passed to the initialize function, which by default is an empty list, in which case the Container object can store objects of mixed and arbitrary types. If the container will only contain objects of one particular type, for example, double values, it will be both more efficient and type safe to initialize the container using this particular type (see Examples section).

R6 constructor

Container$new(x=list())
Container methods

add(elem)  Add elem to Container.
apply(f)   Apply function f to all elements and return results in a list
clear()  Remove all elements from the Container.
discard(elem, right=FALSE)  Search for first elem in Container and, if found, remove it. If right is TRUE, search from right to left.
empty()  Return TRUE if the Container is empty, else FALSE.
has(elem)  Return TRUE if Container contains elem else FALSE.
print(list.len)  Print object representation similar to str
remove(elem, right=FALSE)  Same as discard, but throw an error if not found.
size()   Return size of the Container.
type()  Return type (or mode) of internal vector containing the elements.
values()  Return a copy of all elements in the same format as they are stored in the object.

Author(s)

Roman Pahl

See Also

Iterable, Deque, Set, and Dict

Examples

c0 <- Container$new()
c0$size()       # 0
c0$add(1)
c0$add(2)$add("A")    # chaining example
c0$has(2)       # TRUE
c0$discard(2)$has(2)  # FALSE

## Not run:
c0$remove(2)     # Error : 2 not in Container

## End(Not run)
c0$discard(2)$has(2)  # still FALSE, but no error

# Container types
Container$new(list("A", 1))$type()   # "list"
Container$new(numeric(0))$type()    # "double"
Container$new(0+0i)$type()           # "complex"
Container$new(letters[1:3])$type()   # "character"
Container$new(letters[1:3])$values() # "a" "b" "c"
Container$new(1L)$type()             # "integer"
Container$new(1L)$add(2.3)$values()  # since integer type, equals c(1, 2)
container.pkg

Container, Deque, Set, and Dict (aka Map) - R6 based container classes with iterators and reference semantics.

Description

Implements a general Container class with typical member functions to insert, delete and access objects from the container. The Container class serves as the base class for the Deque, Set and Dict classes (resembling 'Python’s dict type). Supports iterators and, being R6 classes, reference semantics. The focus of implementation was not on speed but to define consistent class interfaces based on a meaningful class hierarchy.

ContainerS3

Container S3 interface

Description

This function creates a container data structure with typical member functions to insert, delete and access objects from the container. It also serves as the base class for objects created with deque, set, and dict.

Usage

container(x = list())
as.container(x)
is.container(x)
add(x, ...)
clear(x)
clone(x, ...)
discard(x, ...)
empty(x)
has(x, ...)
remove(x, ...)
size(x)
Arguments

x  initial elements passed to constructor or object of class Container passed to member methods.

... further arguments

Details

The underlying data structure is based on R vectors (or lists), with the mode being set to the mode (or type) of the value passed to the initialize function, which by default is an empty list, in which case the Container object can store objects of mixed and arbitrary types. If the container will only contain objects of one particular type, for example, double values, it will be both more efficient and type safe to initialize the container using this particular type (see Examples section).

S3 methods for class Container

iter(cont) Create iterator from cont.
add(cont, elem) Add elem to cont.
clear(cont) Remove all elements from the cont.
clone(cont) Create a copy of cont object. For more details see documentation of R6Class.
discard(cont, elem, right=FALSE) Search for first elem in cont and, if found, remove it. If right is TRUE, search from right to left.
empty(cont) Return TRUE if the cont is empty, else FALSE.
has(cont, elem) Return TRUE if cont contains elem else FALSE.
print(cont, list.len, ...) Print container object representation similar to str
remove(cont, elem, right=FALSE) Same as discard, but throw an error if elem does not exist.
size(cont) Return size of the cont.
type(cont) Return type (or mode) of internal vector containing the elements of the container.
values(cont) Return a copy of all elements in the same format as they are stored in the object.

See Also

Container, +.Container.

Examples

c0 <- container(list(2, "A"))
size(c0)  
add(c0, 1)
c0$has(2)  
discard(c0, 2)
has(c0, 2)  

## Not run:
c$remove(2)  # Error : 2 not in Container

## End(Not run)
discard(c$, 2)  # ok (no effect)

type(container(list("A", 1)))  # "list"
type(container(numeric(0)))  # "double"
type(container(0+0i))  # "complex"
type(container(letters[1:3]))  # "character"
values(container(letters[1:3]))  # "a" "b" "c"
type(container(1L))  # "integer"
values(add(container(1L), 2.3))  # since integer type, equals c(1, 2)

---

### ContainerS3op

#### Container operators

**Description**

- Binary operators for Container objects.

**Usage**

```r
## S3 method for class 'Container'
c1 + c2
```

**Arguments**

- `c1`: Container object
- `c2`: Container object

**Details**

- `c1 + c2`: return `c1` and `c2` combined (as a copy)

**Value**

- Container object
**Deque**

---

**Description**

Deques are a generalization of stacks and queues typically with methods to add, remove and access elements at both sides of the underlying data sequence. As such, the Deque can also be used to mimic both stacks and queues.

**Usage**

Deque

**Format**

An object of class `R6ClassGenerator` of length 24.

**Details**

Inherits from `Container` and extends it by `pop` and `peek` methods, element counting, and reverse and rotate functionality.

**Inherited methods**

Inherits all methods from `Container` class.

**R6 constructor**

`Deque$new(x=list())`

**Deque methods**

- `addleft(elem)` Add `elem` to left side of the Deque.
- `count(elem)` Count number of `elem` occurrences.
- `pop()` Remove and return element from the right side of the Deque.
- `popleft()` Remove and return an element from the left side of the Deque.
- `peek()` Peek at last element on the right side without removing it.
- `peekleft()` Peek at first element on the left side without removing it.
- `reverse()` Reverse all elements of the Deque in-place.
- `rotate(n=1)` Rotate the Deque elements `n` steps to the right. If `n` is negative, rotate to the left.

**Author(s)**

Roman Pahl
See Also

Container

Examples

```r
# addleft
d <- Deque$new(1L)$addleft(2)
d$values() # 2 1
Deque$new(0L)$addleft(3:1)$values() # 3 2 1 0

# count
Deque$new(c("Lisa", "Bob", "Bob"))$count("Bob") # 2

# peek and pop
d <- Deque$new(1:3)
d$peek() # 3
d$pop() # 3
d$pop() # 2
d$pop() # 1
"dontrun{
" # Error: pop at empty Deque
"}
Deque$new(1:3)$reverse()$values() # 3 2 1
Deque$new(1:3)$rotate()$values() # 3 1 2
Deque$new(1:3)$rotate(2)$values() # 2 3 1
Deque$new(1:3)$rotate(-1)$values() # 2 3 1
```

Description

Access elements from Deque or Dict objects.

Usage

```r
peek(x, ...)
pop(x, ...)
```

Arguments

- `x` object of class Deque or Dict
- `...` further arguments
Deques are a generalization of stacks and queues typically with methods to add, remove and access elements at both sides of the underlying data sequence. As such, `deque` can also be used to mimic both stacks and simple queues.

**Usage**

- `deque(x = list())`
- `as.deque(x)`
- `is.deque(x)`
- `addleft(x, ...)`
- `count(x, ...)`
- `peekleft(x)`
- `popleft(x)`
- `reverse(x)`
- `rotate(x, ...)`

**Arguments**

- `x` initial elements passed to constructor or object of class Deque passed to member methods.
- `...` further arguments

**Details**

Inherits from `container` and extends it by `pop` and `peek` methods, element counting, and `reverse` and `rotate` functionality.

**S3 methods for Deque objects**

- `addleft(deq, elem)` Add `elem` to left side of the `deq`.
- `count(deq, elem)` Count number of `elem` occurrences.
- `pop(deq)` Remove and return element from the right side of the `deq`.
- `popleft(deq)` Remove and return an element from the left side of the `deq`.
peek(deq)  Peek at last element on the right side without removing it.
peekleft(deq)  Peek at first element on the left side without removing it.
reverse(deq)  Reverse all elements of the deq in-place.
rotate(deq, n=1L)  Rotate the deq elements n steps to the right. If n is negative, rotate to the left.

See Also

container, Deque, .+Deque

Examples

# addleft
d <- 2 + deque(1L)
values(d)  # 2
values(3:1 + deque(0L))  # 3 1 0

# count
count(deque(c("Lisa", "Bob", "Bob", "Bob")))  # 2

# peek and pop
d <- deque(1:3)
peek(d)  # 3
pop(d)  # 3
pop(d)  # 2
pop(d)  # 1

## Not run:
d$pop()  # Error: pop at empty Deque

## End(Not run)

d <- deque(1:3)
print(d)
reverse(d)  # 3 1 2
print(d)

rotate(d)
values(d)  # 1 3 2
values(rotate(d, -1))  # 3 2 1
values(rotate(d, 2))  # 2 1 3
**Dict**

Usage

```r
## S3 method for class 'Deque'
+x1 + x2
```

Arguments

- `x1`: primitive or Deque object
- `x2`: primitive or Deque object

Details

```r
x1 + x2:
```

Value

Deque object

---

Description

The Dict resembles Python’s dict type, and is implemented as a specialized associative (or mapping) Container thus sharing all Container methods with some of them being overridden to account for the associative key-value pair semantic.

Usage

```r
Dict
```

Format

An object of class R6ClassGenerator of length 24.

Inherited methods

Inherits all methods from Container but overrides the internal initialize function and the following member functions:

- `add(key, value)` If key not yet in Dict, insert value at key, otherwise signal an error.
- `discard(key)` If key in Dict, remove it.
- `has(key)` TRUE if key in Dict else FALSE.
- `remove(key)` If key in Dict, remove it, otherwise raise an error.

R6 constructor

```r
Dict$new(x=list())
```
Dict methods

get(key) If key in Dict, return value, else throw key-error.
keys() Return a character vector of all keys.
peek(key, default=NULL) Return the value for key if key is in the Dict, else default.
pop(key) If key in Dict, return a copy of its value and discard it afterwards.
popitem() Remove and return an arbitrary (key, value) pair from the dictionary. popitem() is useful to destructively iterate over a Dict, as often used in set algorithms.
set(key, value, add=FALSE) Like add but overwrites value if key is already in the Dict. If key not in Dict, an error is thrown unless add was set to TRUE
sort(decr=FALSE) Sort values in dictionary according to keys.
update(other=Dict$new()) Adds element(s) of other to the dictionary if the key is not in the dictionary and updates the key with the new value otherwise.

Author(s)
Roman Pahl

See Also
Container

Examples

ages <- Dict$new(c(Peter=24, Lisa=23, Bob=32))
ages$has("Peter") # TRUE
ages$peek("Lisa") # 23
ages$peek("Mike") # NULL
ages$add("Mike", 18)
ages$peek("Mike") # 18
ages$keys()
print(ages)

## Not run:
Dict$new(list(Peter=20))$add("Peter", 22) # key already in Dict
Dict$new(c(Peter=24, Lisa=23, Bob=32, Peter=20)) # Error: duplicated keys

## End(Not run)

dictS3

Dict constructors

Description

The dict resembles Python’s dict type, and is implemented as a specialized associative (or mapping) container thus sharing all container methods with some of them being overridden to account for the associative key-value pair semantic.
dictS3

Usage

dict(x = list())
as.dict(x)
is.dict(x)
getval(x, ...)
keys(x)
popitem(x)
setval(x, ...)
sortkey(x, ...)

Arguments

x initial elements passed to constructor or object of class Dict passed to member methods.
...
further arguments

S3 methods for class Dict

add(dic, key, value) If key not yet in dic, insert value at key, otherwise signal an error.
discard(dic, key) If key in dic, remove it.
has(dic, key) TRUE if key in dic else FALSE.
remove(dic, key) If key in dic, remove it, otherwise raise an error.
getval(dic) If key in dic, return value, else throw key-error.
keys(dic) Return a character vector of all keys.
peek(dic, key, default=NULL) Return the value for key if key is in the dic, else default.
pop(dic, key) If key in dic, return a copy of its value and discard it afterwards.
popitem(dic) Remove and return an arbitrary (key, value) pair from the dictionary. popitem() is useful to destructively iterate over a dic, as often used in set algorithms.
setval(dic, key, value, add=FALSE) Like add but overwrites value if key is already in the dic. If key not in dic, an error is thrown unless add was set to TRUE.
sortkey(dic, decr=FALSE) Sort values in dictionary according to keys.
update(dic, other=dict()) Adds element(s) of other to the dictionary if the key(s) are not in the dictionary and updates all keys with the new value(s) otherwise.

See Also

cntainer, Dict, +.Dict, [<-.Dict, [[<-.Dict, [.Dict, [,..Dict
Examples

```r
dictS3binOp

# Create a dictionary
ages <- dict(c(Peter=24, Lisa=23, Bob=32))
has(ages, "Peter")  # TRUE
ages["Lisa"]  # 23
ages["Mike"]  # NULL
ages["Mike"] <- 18
ages["Mike"]  # 18
keys(ages)
print(ages)

## Not run:
ages["Peter"] <- 24 + 1  # key 'Peter' already in Dict
dict(c(Peter=24, Peter=20))  # Error: duplicated keys

## End(Not run)
```

dictS3binOp  Binary dict operators

Description

Binary operators for Dict objects.

Usage

```r
## S3 method for class 'Dict'
d1 + d2

## S3 method for class 'Dict'
d1 - d2
```

Arguments

- `d1`  
  *Dict* object
- `d2`  
  *Dict* object

Details

- `d1 + d2`: return a copy of `d1` updated by `d2`.
- `d1 - d2`: return a copy of `d1` with all keys being removed that occurred in `d2`.

Value

*Dict* object
**dictS3replace**

**Extract or replace Dict values**

**Description**

Access and assignment operators for Dict objects.

**Usage**

```r
## S3 replacement method for class 'Dict'
dic[[key, add = FALSE]] <- value
```

```r
## S3 replacement method for class 'Dict'
dic[key] <- value
```

```r
## S3 method for class 'Dict'
dic[[key]]
```

```r
## S3 method for class 'Dict'
dic[key, default = NULL]
```

**Arguments**

- `dic` **Dict** object
- `key` (character) the key
- `add` (logical) if TRUE, value is added if not yet in dict. If FALSE and value not yet in dict, an error is signaled.
- `value` the value associated with the key
- `default` the default value

**Details**

- `dic[key] <- value`: If key not yet in dic, insert value at key, otherwise raise an error.
- `dic[key]`: If key in dic, return value, else throw key-error.
- `dic[key, default=NULL]`: Return the value for key if key is in dic, else default.

**Value**

- updated **Dict** object
- value at key
- element found at key, or default if not found.
Description

An `Iterable` is an object that provides an `iter()` method, which is expected to return an `Iterator` object. This class defines the abstract class interface such that each class inheriting this class provides an `iter()` method and must implement a private method `create_iter`, which must return an `Iterator` object.

Usage

`Iterable`

Format

An object of class `R6ClassGenerator` of length 24.

Inherited methods

Inherits method `iter` from abstract `Iterable` class.

Iterable method/interface

`iter()` Return `Iterator` object.

Author(s)

Roman Pahl

See Also

`Iterator` and `Container`

Description

An `Iterator` is an object that allows to iterate over sequences. It implements `_next` and `get` to iterate and retrieve the value of the sequence it is associated with.
**Iterator**

**Usage**

```
Iterator

iter(x)

is_iterator(x)

itbegin(it)

itget(it)

itget_next(it)

itpos(it)

ithas_next(it)

itnext(it)
```

**Arguments**

- `x` : iterable object, e.g., `list`, `vector`, `Container`
- `it` : `Iterator` object

**Format**

An object of class `R6ClassGenerator` of length 24.

**Constructor**

```
Iterator$new(x)
```

**Iterator interface**

- `begin()` : Reset iterator position to 1.
- `get()` : Get value at current iterator position.
- `get_next()` : Get value after incrementing by one.
- `pos()` : Return current iterator position.
- `has_next()` : Return TRUE if there is a next element.
- `next()` : Increment iterator to point at next element.

**S3 method interface**

- `itbegin(it)` : Reset iterator position to 1.
- `itget(it)` : Get value at current iterator position.
- `itget_next()` : Get value after incrementing by one.
Set

A Set class

Description

The Set is considered and implemented as a specialized Container, that is, elements are always unique in the Container and it provides typical set operations such as union and intersect.

Author(s)

Roman Pahl

See Also

Iterable, Container, container

Examples

# Iterator on primitive list
it <- Iterator$new(list("A", 1, 2))
while(it$has_next()) {
  print(it$get_next())
}

# Iterator from Container object
d <- deque(1:3)
it <- iter(d)
sum <- 0
while(it$has_next()) {
  sum <- sum + it$get_next()
}
print(sum)

# S3 method interface
it <- iter(list("A", 1, 2))
while(ithas_next(it)) {
  print(itnext(it))
}

itnext(it) # FALSE
print(it) # <Iterator> at position 3
begin(it)
print(it) # <Iterator> at position 0

ithas_next(it) # FALSE
print(it) # <Iterator> at position 3
begin(it)
print(it) # <Iterator> at position 0

itpos() Return current iterator position.
ithas_next(it) Return TRUE if there is a next element.
itnext(it) Increment iterator to point at next element.
Usage

Set

Format

An object of class R6ClassGenerator of length 24.

R6 constructor

Set$new(x=list())

Inherited methods

Inherits all methods from Container, but overrides add:

add(elem) If not already in set, add elem.

Set methods

union(s) Return new Set as a result of the union of this and s.
intersect(s) Return new Set as a result of the intersection of this and s.
diff(s) Return new Set as a result of the set difference between this and s.
is.subset(s) TRUE if this is a subset of s, else FALSE.
is.superset(s) TRUE if this is a superset of s, else FALSE.

Author(s)

Roman Pahl

See Also

Container

Examples

s1 <- Set$new()$add("A")
s1$values() # "A"
s1$add(2)$add("A")$values() # "A" 2
s1$remove("A")$values() # 2

#' \dontrun{
#' s1$remove(3) # Error: 3 not in Set
#' }
Description

The set is considered and implemented as a specialized container in which elements are always unique. It provides typical set operations such as union (+) and intersect (/).

Usage

set(x = list())
as.set(x)
is.set(x)

Arguments

x (vector or list) initial elements of the set

S3 methods for class Set

add(x, elem) If not already in set x, add elem.

Author(s)

Roman Pahl

See Also

container, Set, +.Set, /.Set, -.Set,<.Set,>.Set

Examples

s1 <- set(list(1, 2, "A", "B"))
s2 <- set(values(s1))
s1 == s2  # TRUE
s1$add(1)  # 1 was already in set, therefore ...
s1 == s2  # ... still TRUE
s1$add(3)
s1 == s2  # FALSE
s1 > s2  # TRUE
s1 - s2  # the added element
unlist(values(s1 / s2))

## Not run:
s1$remove(4)  # Error: 3 not in Set

## End(Not run)
**Description**

Binary operators for Set objects.

**Usage**

```r
## S3 method for class 'Set'
s1 + s2
## S3 method for class 'Set'
s1 / s2
## S3 method for class 'Set'
s1 - s2
## S3 method for class 'Set'
s1 == s2
## S3 method for class 'Set'
s1 < s2
## S3 method for class 'Set'
s1 > s2
```

**Arguments**

- `s1` : Set object
- `s2` : Set object

**Value**

- union of both sets
- intersection of both sets
- set-difference of both sets
- TRUE if both sets are equal, else FALSE
- TRUE if s1 is subset of s2, else FALSE
- TRUE if s1 is superset of s2, else FALSE
Index

+Topic datasets
  Container, 2
  Deque, 7
  Dict, 11
  Iterable, 16
  Iterator, 16
  Set, 18
+ .Container, 5
+ .Container (ContainerS3op), 6
+ .Deque (dequeS3binOp), 10
+ .Dict (dictS3binOp), 14
+ .Set (setS3binOp), 21
-.Dict (dictS3binOp), 14
-.Set (setS3binOp), 21
/.Set, 20
<.Set (setS3binOp), 21
<= .Set (setS3binOp), 21
>.Set, 20
>.Set (setS3binOp), 21
[[.Dict, 13
[[.Dict (dictS3replace), 15
[[<.Dict (dictS3replace), 15
[[[.dict (dictS3replace), 15
[[[<.dict (dictS3replace), 15
add (ContainerS3), 4
addleft (dequeS3), 9
as .container (ContainerS3), 4
as .deque (dequeS3), 9
as .dict (dictS3), 12
as .set (setS3), 20

clear (ContainerS3), 4
close (ContainerS3), 4

Container, 2, 5–8, 11, 12, 16–19
container, 9, 10, 12, 13, 18, 20
container (ContainerS3), 4
container.pkg, 4
container.pkg-package (container.pkg), 4
ContainerS3, 4
ContainerS3op, 6
count, 9
count (dequeS3), 9

Deque, 2, 3, 7, 7, 10, 11
ddeque, 4, 9
ddeque (dequeS3), 9
DequeDictS3funcs, 8
ddequeS3, 9
ddequeS3binOp, 10
Dict, 2, 3, 11, 13–15
dict, 4
dict (dictS3), 12
dictS3, 12
dictS3binOp, 14
dictS3replace, 15
discard (ContainerS3), 4
eempty (ContainerS3), 4
getval (dictS3), 12
has (ContainerS3), 4

is .container (ContainerS3), 4
is .deque (dequeS3), 9
is .dict (dictS3), 12
is .iterator (Iterator), 16
is .set (setS3), 20
itbegin (Iterator), 16
iter (Iterator), 16
Iterable, 3, 16, 16, 18
Iterator, 16, 16, 17
itget (Iterator), 16
itget_next (Iterator), 16
ithas_next (Iterator), 16
itnext (Iterator), 16
itpos (Iterator), 16

keys (dictS3), 12

list, 17
peek, 9
peek (DequeDictS3funcs), 8
peekleft (dequeS3), 9
pop, 9
pop (DequeDictS3funcs), 8
popitem (dictS3), 12
popleft (dequeS3), 9

R6Class, 5
remove (ContainerS3), 4
reverse, 9
reverse (dequeS3), 9
rotate, 9
rotate (dequeS3), 9

Set, 2, 3, 18, 20, 21
set, 4
set (setS3), 20
setS3, 20
setS3binOp, 21
setval (dictS3), 12
size (ContainerS3), 4
sortkey (dictS3), 12
str, 3, 5

type (ContainerS3), 4

values (ContainerS3), 4
vector, 17