Module java.base
Package java.util.concurrent

Class CopyOnWriteArrayList<E>

java.lang.Object
    java.util.concurrent.CopyOnWriteArrayList<E>

Type Parameters:

E - the type of elements held in this list

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

---

```
public class CopyOnWriteArrayList<E>
extends Object
implements List<E>, RandomAccess, Cloneable, Serializable
```

A thread-safe variant of ArrayList in which all mutative operations (add, set, and so on) are implemented by making a fresh copy of the underlying array.

This is ordinarily too costly, but may be *more* efficient than alternatives when traversal operations vastly outnumber mutations, and is useful when you cannot or don't want to synchronize traversals, yet need to preclude interference among concurrent threads. The "snapshot" style iterator method uses a reference to the state of the array at the point that the iterator was created. This array never changes during the lifetime of the iterator, so interference is impossible and the iterator is guaranteed not to throw ConcurrentModificationException. The iterator will not reflect additions, removals, or changes to the list since the iterator was created. Element-changing operations on iterators themselves (remove, set, and add) are not supported. These methods throw UnsupportedOperationException.

All elements are permitted, including null.

Memory consistency effects: As with other concurrent collections, actions in a thread prior to placing an object into a CopyOnWriteArrayList *happen-before* actions subsequent to the access or removal of that element from the CopyOnWriteArrayList in another thread.

This class is a member of the Java Collections Framework.

Since:

1.5

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor	Description
CopyOnWriteArrayList()	Creates an empty list.

`CopyOnWriteArrayList(E[] toCopyIn)`	Creates a list holding a copy of the given array.
`CopyOnWriteArrayList(Collection<? extends E> c)`	Creates a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
void	`add(int index, E element)`	Inserts the specified element at the specified position in this list.
boolean	`add(E e)`	Appends the specified element to the end of this list.
boolean	`addAll(int index, Collection<? extends E> c)`	Inserts all of the elements in the specified collection into this list, starting at the specified position.
boolean	`addAll(Collection<? extends E> c)`	Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
int	`addAllAbsent(Collection<? extends E> c)`	Appends all of the elements in the specified collection that are not already contained in this list, to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean	`addIfAbsent(E e)`	Appends the element, if not present.
void	`clear()`	Removes all of the elements from this list.
Object	`clone()`	Returns a shallow copy of this list.
boolean	`contains(Object o)`	Returns `true` if this list contains the specified element.
boolean	`containsAll(Collection<?> c)`	Returns `true` if this list contains all of the elements of the

		specified collection.
boolean	**equals**(**Object** o)	Compares the specified object with this list for equality.
void	**forEach**(**Consumer**<? super **E**> action)	Performs the given action for each element of the `Iterable` until all elements have been processed or the action throws an exception.
E	**get**(int index)	Returns the element at the specified position in this list.
int	**hashCode**()	Returns the hash code value for this list.
int	**indexOf**(**E** e, int index)	Returns the index of the first occurrence of the specified element in this list, searching forwards from `index`, or returns -1 if the element is not found.
int	**indexOf**(**Object** o)	Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	**isEmpty**()	Returns `true` if this list contains no elements.
Iterator<**E**>	**iterator**()	Returns an iterator over the elements in this list in proper sequence.
int	**lastIndexOf**(**E** e, int index)	Returns the index of the last occurrence of the specified element in this list, searching backwards from `index`, or returns -1 if the element is not found.
int	**lastIndexOf**(**Object** o)	Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
ListIterator<**E**>	**listIterator**()	Returns a list iterator over the elements in this list (in proper sequence).
ListIterator<**E**>	**listIterator**(int index)	Returns a list iterator over the elements in this list (in proper

		sequence), starting at the specified position in the list.
E	**remove**(int index)	Removes the element at the specified position in this list.
boolean	**remove**(**Object** o)	Removes the first occurrence of the specified element from this list, if it is present.
boolean	**removeAll**(**Collection**<?> c)	Removes from this list all of its elements that are contained in the specified collection.
boolean	**removeIf**(**Predicate**<? super **E**> filter)	Removes all of the elements of this collection that satisfy the given predicate.
boolean	**retainAll**(**Collection**<?> c)	Retains only the elements in this list that are contained in the specified collection.
E	**set**(int index, **E** element)	Replaces the element at the specified position in this list with the specified element.
int	**size**()	Returns the number of elements in this list.
Spliterator<**E**>	**spliterator**()	Returns a `Spliterator` over the elements in this list.
List<**E**>	**subList**(int fromIndex, int toIndex)	Returns a view of the portion of this list between `fromIndex`, inclusive, and `toIndex`, exclusive.
Object[]	**toArray**()	Returns an array containing all of the elements in this list in proper sequence (from first to last element).
<T> T[]	**toArray**(T[] a)	Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.
String	**toString**()	Returns a string representation of this list.

Methods declared in class java.lang.Object

`finalize`, `getClass`, `notify`, `notifyAll`, `wait`, `wait`, `wait`

Methods declared in interface java.util.Collection

parallelStream, stream, toArray

Methods declared in interface java.util.List

replaceAll, sort

Constructor Details

CopyOnWriteArrayList

```
public CopyOnWriteArrayList()
```

Creates an empty list.

CopyOnWriteArrayList

```
public CopyOnWriteArrayList(Collection<? extends E> c)
```

Creates a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Parameters:

c - the collection of initially held elements

Throws:

NullPointerException - if the specified collection is null

CopyOnWriteArrayList

```
public CopyOnWriteArrayList(E[] toCopyIn)
```

Creates a list holding a copy of the given array.

Parameters:

toCopyIn - the array (a copy of this array is used as the internal array)

Throws:

NullPointerException - if the specified array is null

Method Details

size

```
public int size()
```

Returns the number of elements in this list.

Specified by:

`size` in interface `Collection<E>`

Specified by:

`size` in interface `List<E>`

Returns:

the number of elements in this list

isEmpty

```
public boolean isEmpty()
```

Returns `true` if this list contains no elements.

Specified by:

`isEmpty` in interface `Collection<E>`

Specified by:

`isEmpty` in interface `List<E>`

Returns:

`true` if this list contains no elements

contains

```
public boolean contains(Object o)
```

Returns `true` if this list contains the specified element. More formally, returns `true` if and only if this list contains at least one element `e` such that `Objects.equals(o, e)`.

Specified by:

`contains` in interface `Collection<E>`

Specified by:

`contains` in interface `List<E>`

Parameters:

`o` - element whose presence in this list is to be tested

Returns:

`true` if this list contains the specified element

indexOf

```
public int indexOf(Object o)
```

Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. More formally, returns the lowest index `i` such that `Objects.equals(o, get(i))`, or -1 if there is no such index.

Specified by:

`indexOf` in interface `List<E>`

Parameters:

`o` - element to search for

Returns:

the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element

indexOf

```
public int indexOf(E e,
                   int index)
```

Returns the index of the first occurrence of the specified element in this list, searching forwards from `index`, or returns -1 if the element is not found. More formally, returns the lowest index `i` such that `i >= index && Objects.equals(get(i), e)`, or -1 if there is no such index.

Parameters:

`e` - element to search for

`index` - index to start searching from

Returns:

the index of the first occurrence of the element in this list at position `index` or later in the list; `-1` if the element is not found.

Throws:

`IndexOutOfBoundsException` - if the specified index is negative

lastIndexOf

```
public int lastIndexOf(Object o)
```

Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element. More formally, returns the highest index `i` such that `Objects.equals(o, get(i))`, or -1 if there is no such index.

Specified by:

`lastIndexOf` in interface `List<E>`

Parameters:

`o` - element to search for

Returns:

the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element

lastIndexOf

```
public int lastIndexOf(E e,
                       int index)
```

Returns the index of the last occurrence of the specified element in this list, searching backwards from `index`, or returns -1 if the element is not found. More formally, returns the highest index `i` such that `i <= index && Objects.equals(get(i), e)`, or -1 if there is no such index.

Parameters:

`e` - element to search for

`index` - index to start searching backwards from

Returns:

the index of the last occurrence of the element at position less than or equal to `index` in this list; -1 if the element is not found.

Throws:

`IndexOutOfBoundsException` - if the specified index is greater than or equal to the current size of this list

clone

```
public Object clone()
```

Returns a shallow copy of this list. (The elements themselves are not copied.)

Overrides:

`clone` in class `Object`

Returns:

a clone of this list

See Also:

`Cloneable`

toArray

```
public Object[] toArray()
```

Returns an array containing all of the elements in this list in proper sequence (from first to last element).

The returned array will be "safe" in that no references to it are maintained by this list. (In other words, this method must allocate a new array). The caller is thus free to modify the returned array.

This method acts as bridge between array-based and collection-based APIs.

Specified by:

`toArray` in interface `Collection<E>`

Specified by:

`toArray` in interface `List<E>`

Returns:

an array containing all the elements in this list

See Also:

`Arrays.asList(Object[])`

toArray

```
public <T> T[] toArray(T[] a)
```

Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array. If the list fits in the specified array, it is returned therein. Otherwise, a new array is allocated with the runtime type of the specified array and the size of this list.

If this list fits in the specified array with room to spare (i.e., the array has more elements than this list), the element in the array immediately following the end of the list is set to `null`. (This is useful in determining the length of this list *only* if the caller knows that this list does not contain any null elements.)

Like the `toArray()` method, this method acts as bridge between array-based and collection-based APIs. Further, this method allows precise control over the runtime type of the output array, and may, under certain circumstances, be used to save allocation costs.

Suppose `x` is a list known to contain only strings. The following code can be used to dump the list into a newly allocated array of `String`:

```
 String[] y = x.toArray(new String[0]);
```

Note that `toArray(new Object[0])` is identical in function to `toArray()`.

Specified by:

`toArray` in interface `Collection<E>`

Specified by:

`toArray` in interface `List<E>`

Type Parameters:

T - the component type of the array to contain the collection

Parameters:

a - the array into which the elements of the list are to be stored, if it is big enough; otherwise, a new array of the same runtime type is allocated for this purpose.

Returns:

an array containing all the elements in this list

Throws:

`ArrayStoreException` - if the runtime type of the specified array is not a supertype of the runtime type of every element in this list

`NullPointerException` - if the specified array is null

get

```
public E get(int index)
```

Returns the element at the specified position in this list.

Specified by:

`get` in interface `List<E>`

Parameters:

`index` - index of the element to return

Returns:

the element at the specified position in this list

Throws:

`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index >= size()`)

set

```
public E set(int index,
             E element)
```

Replaces the element at the specified position in this list with the specified element.

Specified by:

`set` in interface `List<E>`

Parameters:

`index` - index of the element to replace

`element` - element to be stored at the specified position

Returns:

the element previously at the specified position

Throws:

`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index >= size()`)

add

```
public boolean add(E e)
```

Appends the specified element to the end of this list.

Specified by:

`add` in interface `Collection<E>`

Specified by:

`add` in interface `List<E>`

Parameters:

`e` - element to be appended to this list

Returns:

`true` (as specified by `Collection.add(E)`)

add

```
public void add(int index,
                E element)
```

Inserts the specified element at the specified position in this list. Shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices).

Specified by:

`add` in interface `List<E>`

Parameters:

`index` - index at which the specified element is to be inserted

`element` - element to be inserted

Throws:

`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index > size()`)

remove

```
public E remove(int index)
```

Removes the element at the specified position in this list. Shifts any subsequent elements to the left (subtracts one from their indices). Returns the element that was removed from the list.

Specified by:

`remove` in interface `List<E>`

Parameters:

`index` - the index of the element to be removed

Returns:

the element previously at the specified position

Throws:

`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index >= size()`)

remove

```
public boolean remove(Object o)
```

Removes the first occurrence of the specified element from this list, if it is present. If this list does not contain the element, it is unchanged. More formally, removes the

element with the lowest index `i` such that `Objects.equals(o, get(i))` (if such an element exists). Returns `true` if this list contained the specified element (or equivalently, if this list changed as a result of the call).

Specified by:

`remove` in interface `Collection<E>`

Specified by:

`remove` in interface `List<E>`

Parameters:

`o` - element to be removed from this list, if present

Returns:

`true` if this list contained the specified element

addIfAbsent

```
public boolean addIfAbsent(E e)
```

Appends the element, if not present.

Parameters:

`e` - element to be added to this list, if absent

Returns:

`true` if the element was added

containsAll

```
public boolean containsAll(Collection<?> c)
```

Returns `true` if this list contains all of the elements of the specified collection.

Specified by:

`containsAll` in interface `Collection<E>`

Specified by:

`containsAll` in interface `List<E>`

Parameters:

`c` - collection to be checked for containment in this list

Returns:

`true` if this list contains all of the elements of the specified collection

Throws:

`NullPointerException` - if the specified collection is null

See Also:

`contains(Object)`

removeAll

`public boolean removeAll(Collection<?> c)`

Removes from this list all of its elements that are contained in the specified collection. This is a particularly expensive operation in this class because of the need for an internal temporary array.

Specified by:

`removeAll` in interface `Collection<E>`

Specified by:

`removeAll` in interface `List<E>`

Parameters:

`c` - collection containing elements to be removed from this list

Returns:

`true` if this list changed as a result of the call

Throws:

`ClassCastException` - if the class of an element of this list is incompatible with the specified collection (optional)

`NullPointerException` - if this list contains a null element and the specified collection does not permit null elements (optional), or if the specified collection is null

See Also:

`remove(Object)`

retainAll

`public boolean retainAll(Collection<?> c)`

Retains only the elements in this list that are contained in the specified collection. In other words, removes from this list all of its elements that are not contained in the specified collection.

Specified by:

`retainAll` in interface `Collection<E>`

Specified by:

`retainAll` in interface `List<E>`

Parameters:

`c` - collection containing elements to be retained in this list

Returns:

`true` if this list changed as a result of the call

Throws:

`ClassCastException` - if the class of an element of this list is incompatible with the specified collection (optional)

`NullPointerException` - if this list contains a null element and the specified collection does not permit null elements (optional), or if the specified collection is null

See Also:

remove(Object)

addAllAbsent

```
public int addAllAbsent(Collection<? extends E> c)
```

Appends all of the elements in the specified collection that are not already contained in this list, to the end of this list, in the order that they are returned by the specified collection's iterator.

Parameters:

c - collection containing elements to be added to this list

Returns:

the number of elements added

Throws:

NullPointerException - if the specified collection is null

See Also:

addIfAbsent(Object)

clear

```
public void clear()
```

Removes all of the elements from this list. The list will be empty after this call returns.

Specified by:

clear in interface Collection<E>

Specified by:

clear in interface List<E>

addAll

```
public boolean addAll(Collection<? extends E> c)
```

Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.

Specified by:

addAll in interface Collection<E>

Specified by:

addAll in interface List<E>

Parameters:

c - collection containing elements to be added to this list

Returns:

true if this list changed as a result of the call

Throws:

`NullPointerException` - if the specified collection is null

See Also:

`add(Object)`

addAll

```
public boolean addAll(int index,
                      Collection<? extends E> c)
```

Inserts all of the elements in the specified collection into this list, starting at the specified position. Shifts the element currently at that position (if any) and any subsequent elements to the right (increases their indices). The new elements will appear in this list in the order that they are returned by the specified collection's iterator.

Specified by:

`addAll` in interface `List<E>`

Parameters:

`index` - index at which to insert the first element from the specified collection

`c` - collection containing elements to be added to this list

Returns:

`true` if this list changed as a result of the call

Throws:

`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index > size()`)

`NullPointerException` - if the specified collection is null

See Also:

`add(int,Object)`

forEach

```
public void forEach(Consumer<? super E> action)
```

Description copied from interface: `Iterable`

Performs the given action for each element of the `Iterable` until all elements have been processed or the action throws an exception. Actions are performed in the order of iteration, if that order is specified. Exceptions thrown by the action are relayed to the caller.

The behavior of this method is unspecified if the action performs side-effects that modify the underlying source of elements, unless an overriding class has specified a concurrent modification policy.

Specified by:

`forEach` in interface `Iterable<E>`

Parameters:

`action` - The action to be performed for each element

Throws:

`NullPointerException` - if the specified action is null

removeIf

```
public boolean removeIf(Predicate<? super E> filter)
```

Description copied from interface: `Collection`

Removes all of the elements of this collection that satisfy the given predicate. Errors or runtime exceptions thrown during iteration or by the predicate are relayed to the caller.

Specified by:

`removeIf` in interface `Collection<E>`

Parameters:

`filter` - a predicate which returns `true` for elements to be removed

Returns:

`true` if any elements were removed

Throws:

`NullPointerException` - if the specified filter is null

toString

```
public String toString()
```

Returns a string representation of this list. The string representation consists of the string representations of the list's elements in the order they are returned by its iterator, enclosed in square brackets (`"[]"`). Adjacent elements are separated by the characters `", "` (comma and space). Elements are converted to strings as by `String.valueOf(Object)`.

Overrides:

`toString` in class `Object`

Returns:

a string representation of this list

equals

```
public boolean equals(Object o)
```

Compares the specified object with this list for equality. Returns `true` if the specified object is the same object as this object, or if it is also a `List` and the sequence of elements returned by an iterator over the specified list is the same as the sequence returned by an iterator over this list. The two sequences are considered to be the same if they have the same length and corresponding elements at the same position in the sequence are *equal*. Two elements `e1` and `e2` are considered *equal* if `Objects.equals(e1, e2)`.

Specified by:

`equals` in interface `Collection<E>`

Specified by:

`equals` in interface `List<E>`

Overrides:

`equals` in class `Object`

Parameters:

`o` - the object to be compared for equality with this list

Returns:

`true` if the specified object is equal to this list

See Also:

`Object.hashCode()`, `HashMap`

hashCode

```
public int hashCode()
```

Returns the hash code value for this list.

This implementation uses the definition in `List.hashCode()`.

Specified by:

`hashCode` in interface `Collection<E>`

Specified by:

`hashCode` in interface `List<E>`

Overrides:

`hashCode` in class `Object`

Returns:

the hash code value for this list

See Also:

`Object.equals(java.lang.Object)`,
`System.identityHashCode(java.lang.Object)`

iterator

```
public Iterator<E> iterator()
```

Returns an iterator over the elements in this list in proper sequence.

The returned iterator provides a snapshot of the state of the list when the iterator was constructed. No synchronization is needed while traversing the iterator. The iterator does *NOT* support the `remove` method.

Specified by:

`iterator` in interface `Collection<E>`

Specified by:

`iterator` in interface `Iterable<E>`

Specified by:

`iterator` in interface `List<E>`

Returns:

an iterator over the elements in this list in proper sequence

listIterator

```
public ListIterator<E> listIterator()
```

Returns a list iterator over the elements in this list (in proper sequence).

The returned iterator provides a snapshot of the state of the list when the iterator was constructed. No synchronization is needed while traversing the iterator. The iterator does *NOT* support the `remove`, `set` or `add` methods.

Specified by:

`listIterator` in interface `List<E>`

Returns:

a list iterator over the elements in this list (in proper sequence)

listIterator

```
public ListIterator<E> listIterator(int index)
```

Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list. The specified index indicates the first element that would be returned by an initial call to `next`. An initial call to `previous` would return the element with the specified index minus one.

The returned iterator provides a snapshot of the state of the list when the iterator was constructed. No synchronization is needed while traversing the iterator. The iterator does *NOT* support the `remove`, `set` or `add` methods.

Specified by:

`listIterator` in interface `List<E>`

Parameters:

`index` - index of the first element to be returned from the list iterator (by a call to `next`)

Returns:

a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list

Throws:

`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index > size()`)

spliterator

```
public Spliterator<E> spliterator()
```

Returns a `Spliterator` over the elements in this list.

The `Spliterator` reports `Spliterator.IMMUTABLE`, `Spliterator.ORDERED`, `Spliterator.SIZED`, and `Spliterator.SUBSIZED`.

The spliterator provides a snapshot of the state of the list when the spliterator was constructed. No synchronization is needed while operating on the spliterator.

Specified by:

`spliterator` in interface `Collection<E>`

Specified by:

`spliterator` in interface `Iterable<E>`

Specified by:

`spliterator` in interface `List<E>`

Returns:

a `Spliterator` over the elements in this list

Since:

1.8

subList

```
public List<E> subList(int fromIndex,
                       int toIndex)
```

Returns a view of the portion of this list between `fromIndex`, inclusive, and `toIndex`, exclusive. The returned list is backed by this list, so changes in the returned list are reflected in this list.

The semantics of the list returned by this method become undefined if the backing list (i.e., this list) is modified in any way other than via the returned list.

Specified by:

`subList` in interface `List<E>`

Parameters:

`fromIndex` - low endpoint (inclusive) of the subList

`toIndex` - high endpoint (exclusive) of the subList

Returns:

a view of the specified range within this list

Throws:

`IndexOutOfBoundsException` - for an illegal endpoint index value (`fromIndex < 0 || toIndex > size || fromIndex > toIndex`)

---

examples. Other versions.