Package org.springframework.util

Class CollectionUtils

java.lang.Object
    org.springframework.util.CollectionUtils

```
public abstract class CollectionUtils
extends Object
```

Miscellaneous collection utility methods. Mainly for internal use within the framework.

Since:

1.1.3

Author:

Juergen Hoeller, Rob Harrop, Arjen Poutsma

Constructor Summary

Constructors

Constructor	Description
`CollectionUtils()`	

Method Summary

All Methods Static Methods Concrete Methods

Modifier and Type	Method	Description
`static List<?>`	`arrayToList(Object source)`	Convert the supplied array into a List.
`static <K,V> Map<K,V>`	`compositeMap(Map<K,V> first, Map<K,V> second)`	Return a (partially unmodifiable) map that combines the provided two maps.
`static <K,V> Map<K,V>`	`compositeMap(Map<K,V> first, Map<K,V> second, BiFunction<K,V,V> putFunction, Consumer<Map<K,V>> putAllFunction)`	Return a map that combines the provided maps.
`static boolean`	`contains(Enumeration<?> enumeration, Object element)`	Check whether the given Enumeration contains the given element.
`static boolean`	`contains(Iterator<?> iterator, Object element)`	Check whether the given Iterator contains the given element.

static boolean	containsAny(Collection <?> source, Collection <?> candidates)	Return true if any element in 'candidates' is contained in 'source'; otherwise returns false.
static boolean	containsInstance(Collection <?> collection, Object element)	Check whether the given Collection contains the given element instance.
static Class <?>	findCommonElementType (Collection <?> collection)	Find the common element type of the given Collection, if any.
static <E> E	findFirstMatch(Collection <?> source, Collection <E> candidates)	Return the first element in 'candidates' that is contained in 'source'.
static Object	findValueOfType(Collection <?> collection, Class <?> [] types)	Find a single value of one of the given types in the given Collection: searching the Collection for a value of the first type, then searching for a value of the second type, etc.
static <T> T	findValueOfType(Collection <?> collection, Class <T> type)	Find a single value of the given type in the given Collection.
static <T> T	firstElement(List <T> list)	Retrieve the first element of the given List, accessing the zero index.
static <T> T	firstElement(Set <T> set)	Retrieve the first element of the given Set, using SortedSet.first() or otherwise using the iterator.
static boolean	hasUniqueObject(Collection <?> collection)	Determine whether the given Collection only contains a single unique object.
static boolean	isEmpty(Collection <?> collection)	Return true if the supplied Collection is null or empty.
static boolean	isEmpty(Map <?,?> map)	Return true if the supplied Map is null or empty.
static <T> T	lastElement(List <T> list)	Retrieve the last element of the given List, accessing the highest index.
static <T> T	lastElement(Set <T> set)	Retrieve the last element of the given Set, using SortedSet.last() or otherwise iterating over all elements (assuming a linked set).

static <E> void	mergeArrayIntoCollection(Object array, Collection<E> collection)	Merge the given array into the given Collection.
static <K,V> void	mergePropertiesIntoMap(Properties props, Map<K,V> map)	Merge the given Properties instance into the given Map, copying all properties (key-value pairs) over.
static <K,V> HashMap<K,V>	newHashMap(int expectedSize)	Instantiate a new HashMap with an initial capacity that can accommodate the specified number of elements without any immediate resize/rehash operations to be expected.
static <E> HashSet<E>	newHashSet(int expectedSize)	Instantiate a new HashSet with an initial capacity that can accommodate the specified number of elements without any immediate resize/rehash operations to be expected.
static <K,V> LinkedHashMap<K,V>	newLinkedHashMap(int expectedSize)	Instantiate a new LinkedHashMap with an initial capacity that can accommodate the specified number of elements without any immediate resize/rehash operations to be expected.
static <E> LinkedHashSet<E>	newLinkedHashSet(int expectedSize)	Instantiate a new LinkedHashSet with an initial capacity that can accommodate the specified number of elements without any immediate resize/rehash operations to be expected.
static <A,E extends A> A[]	toArray(Enumeration<E> enumeration, A[] array)	Marshal the elements from the given enumeration into an array of the given type.
static <E> Iterator<E>	toIterator(Enumeration<E> enumeration)	Adapt an Enumeration to an Iterator.
static <K,V> MultiValueMap<K,V>	toMultiValueMap(Map<K,List<V>> targetMap)	Adapt a Map<K, List<V>> to an MultiValueMap<K, V>.
static <K,V> MultiValueMap<K,V>	unmodifiableMultiValueMap(MultiValueMap<? extends K,? extends V> targetMap)	Return an unmodifiable view of the specified multi-value map.

Methods inherited from class java.lang.Object

`clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait` , `wait` , `wait`

Constructor Details

CollectionUtils

```
public CollectionUtils()
```

Method Details

isEmpty

```
@Contract("null -> true")
public static boolean isEmpty(@Nullable
                              Collection <?> collection)
```

Return `true` if the supplied Collection is `null` or empty. Otherwise, return `false`.

Parameters:

`collection` - the Collection to check

Returns:

whether the given Collection is empty

isEmpty

```
@Contract("null -> true")
public static boolean isEmpty(@Nullable
                              Map <?,?> map)
```

Return `true` if the supplied Map is `null` or empty. Otherwise, return `false`.

Parameters:

`map` - the Map to check

Returns:

whether the given Map is empty

newHashMap

```
public static <K,V> HashMap <K,V> newHashMap(int expectedSize)
```

Instantiate a new `HashMap` with an initial capacity that can accommodate the specified number of elements without any immediate resize/rehash operations to be expected.

is differs from the regular `HashMap` constructor which takes an initial capacity relative to a load actor but is effectively aligned with the JDK's `ConcurrentHashMap(int)` .

Parameters:

`expectedSize` - the expected number of elements (with a corresponding capacity to be derived so that no resize/rehash operations are needed)

Since:

5.3

See Also:

`newLinkedHashMap(int)`

newLinkedHashMap

```
public static <K,V> LinkedHashMap <K,V> newLinkedHashMap(int expectedSize)
```

Instantiate a new `LinkedHashMap` with an initial capacity that can accommodate the specified number of elements without any immediate resize/rehash operations to be expected.

This differs from the regular `LinkedHashMap` constructor which takes an initial capacity relative to a load factor but is aligned with Spring's own `LinkedCaseInsensitiveMap` and `LinkedMultiValueMap` constructor semantics.

Parameters:

`expectedSize` - the expected number of elements (with a corresponding capacity to be derived so that no resize/rehash operations are needed)

Since:

5.3

See Also:

`newHashMap(int)`

newHashSet

```
public static <E> HashSet <E> newHashSet(int expectedSize)
```

Instantiate a new `HashSet` with an initial capacity that can accommodate the specified number of elements without any immediate resize/rehash operations to be expected.

Parameters:

`expectedSize` - the expected number of elements (with a corresponding capacity to be derived so that no resize/rehash operations are needed)

Since:

6.2

See Also:

`newLinkedHashSet(int)`

newLinkedHashSet

```
public static <E> LinkedHashSet <E> newLinkedHashSet(int expectedSize)
```

Instantiate a new `LinkedHashSet` with an initial capacity that can accommodate the specified number of elements without any immediate resize/rehash operations to be expected.

Parameters:

`expectedSize` - the expected number of elements (with a corresponding capacity to be derived so that no resize/rehash operations are needed)

Since:

6.2

See Also:

`newHashSet(int)`

arrayToList

```
public static List<?> arrayToList(@Nullable
                                  Object source)
```

Convert the supplied array into a List. A primitive array gets converted into a List of the appropriate wrapper type.

NOTE: Generally prefer the standard `Arrays.asList(T...)` method. This `arrayToList` method is just meant to deal with an incoming Object value that might be an `Object[]` or a primitive array at runtime.

A `null` source value will be converted to an empty List.

Parameters:

`source` - the (potentially primitive) array

Returns:

the converted List result

See Also:

`ObjectUtils.toObjectArray(Object)`,
`Arrays.asList(Object[])`

mergeArrayIntoCollection

```
public static <E> void mergeArrayIntoCollection(@Nullable
                                                Object array,
                                                Collection<E> collection)
```

Merge the given array into the given Collection.

Parameters:

`array` - the array to merge (may be `null`)

`collection` - the target Collection to merge the array into

mergePropertiesIntoMap

```
public static <K,V> void mergePropertiesIntoMap(@Nullable
                                                Properties props,
                                                Map<K,V> map)
```

Merge the given Properties instance into the given Map, copying all properties (key-value pairs) over.

Uses `Properties.propertyNames()` to even catch default properties linked into the original Properties instance.

Parameters:

`props` - the Properties instance to merge (may be `null`)

`map` - the target Map to merge the properties into

contains

```
public static boolean contains(@Nullable
                               Iterator<?> iterator,
                               Object element)
```

Check whether the given Iterator contains the given element.

Parameters:

`iterator` - the Iterator to check

`element` - the element to look for

Returns:

`true` if found, `false` otherwise

contains

```
public static boolean contains(@Nullable
                               Enumeration<?> enumeration,
                               Object element)
```

Check whether the given Enumeration contains the given element.

Parameters:

`enumeration` - the Enumeration to check

`element` - the element to look for

Returns:

`true` if found, `false` otherwise

containsInstance

```
public static boolean containsInstance(@Nullable
                                       Collection<?> collection,
                                       Object element)
```

eck whether the given Collection contains the given element instance.

Enforces the given instance to be present, rather than returning `true` for an equal element as well.

Parameters:

`collection` - the Collection to check

`element` - the element to look for

Returns:

`true` if found, `false` otherwise

containsAny

```
public static  boolean  containsAny(Collection <?>  source,
                                   Collection <?>  candidates)
```

Return `true` if any element in '`candidates`' is contained in '`source`'; otherwise returns `false`.

Parameters:

`source` - the source Collection

`candidates` - the candidates to search for

Returns:

whether any of the candidates has been found

findFirstMatch

```
@Nullable
public static  <E>  E  findFirstMatch(Collection <?>  source,
                                      Collection <E>  candidates)
```

Return the first element in '`candidates`' that is contained in '`source`'. If no element in '`candidates`' is present in '`source`' returns `null`. Iteration order is `Collection` implementation specific.

Parameters:

`source` - the source Collection

`candidates` - the candidates to search for

Returns:

the first present object, or `null` if not found

findValueOfType

```
@Nullable
public static  <T>  T  findValueOfType(Collection <?>  collection,
                                       @Nullable
                                       Class <T>  type)
```

Find a single value of the given type in the given Collection.

Parameters:

`llection` - the Collection to search

`ype` - the type to look for

Returns:

a value of the given type found if there is a clear match, or `null` if none or more than one such value found

findValueOfType

```
@Nullable
public static Object findValueOfType(Collection<?> collection,
                                     Class<?>[] types)
```

Find a single value of one of the given types in the given Collection: searching the Collection for a value of the first type, then searching for a value of the second type, etc.

Parameters:

`collection` - the collection to search

`types` - the types to look for, in prioritized order

Returns:

a value of one of the given types found if there is a clear match, or `null` if none or more than one such value found

hasUniqueObject

```
public static boolean hasUniqueObject(Collection<?> collection)
```

Determine whether the given Collection only contains a single unique object.

Parameters:

`collection` - the Collection to check

Returns:

`true` if the collection contains a single reference or multiple references to the same instance, `false` otherwise

findCommonElementType

```
@Nullable
public static Class<?> findCommonElementType(Collection<?> collection)
```

Find the common element type of the given Collection, if any.

Parameters:

`collection` - the Collection to check

Returns:

the common element type, or `null` if no clear common type has been found (or the collection was empty)

firstElement

```
@Nullable
public static <T> T firstElement(@Nullable
                                 Set<T> set)
```

Retrieve the first element of the given Set, using `SortedSet.first()` or otherwise using the iterator.

Parameters:

`set` - the Set to check (may be `null` or empty)

Returns:

the first element, or `null` if none

Since:

5.2.3

See Also:

`SortedSet`, `LinkedHashMap.keySet()`, `LinkedHashSet`

firstElement

```
@Nullable
public static <T> T firstElement(@Nullable
                                 List<T> list)
```

Retrieve the first element of the given List, accessing the zero index.

Parameters:

`list` - the List to check (may be `null` or empty)

Returns:

the first element, or `null` if none

Since:

5.2.3

lastElement

```
@Nullable
public static <T> T lastElement(@Nullable
                                Set<T> set)
```

Retrieve the last element of the given Set, using `SortedSet.last()` or otherwise iterating over all elements (assuming a linked set).

Parameters:

`set` - the Set to check (may be `null` or empty)

Returns:

the last element, or `null` if none

Since:

5.0.3

ee Also:

`rtedSet`, `LinkedHashMap.keySet()`, `LinkedHashSet`

lastElement

```
@Nullable
public static <T> T lastElement(@Nullable
                                List<T> list)
```

Retrieve the last element of the given List, accessing the highest index.

Parameters:

`list` - the List to check (may be `null` or empty)

Returns:

the last element, or `null` if none

Since:

5.0.3

toArray

```
public static <A,E extends A> A[] toArray(Enumeration<E> enumeration,
                                          A[] array)
```

Marshal the elements from the given enumeration into an array of the given type. Enumeration elements must be assignable to the type of the given array. The array returned will be a different instance than the array given.

toIterator

```
public static <E> Iterator<E> toIterator(@Nullable
                                         Enumeration<E> enumeration)
```

Adapt an `Enumeration` to an `Iterator`.

Parameters:

`enumeration` - the original `Enumeration`

Returns:

the adapted `Iterator`

toMultiValueMap

```
public static <K,V> MultiValueMap<K,V> toMultiValueMap(Map<K,List<V>> targetMap)
```

Adapt a `Map<K, List<V>>` to an `MultiValueMap<K, V>`.

Parameters:

`targetMap` - the original map

Returns:

the adapted multi-value map (wrapping the original map)

ce:

3.1

unmodifiableMultiValueMap

```
public static <K,V> MultiValueMap<K,V> unmodifiableMultiValueMap
(MultiValueMap<? extends K,? extends V> targetMap)
```

Return an unmodifiable view of the specified multi-value map.

Parameters:

`targetMap` - the map for which an unmodifiable view is to be returned.

Returns:

an unmodifiable view of the specified multi-value map

Since:

3.1

compositeMap

```
public static <K,V> Map<K,V> compositeMap(Map<K,V> first,
                                          Map<K,V> second)
```

Return a (partially unmodifiable) map that combines the provided two maps. Invoking `Map.put(Object, Object)` or `Map.putAll(Map)` on the returned map results in an `UnsupportedOperationException`.

In the case of a key collision, `first` takes precedence over `second`. In other words, entries in `second` with a key that is also mapped by `first` are effectively ignored.

Parameters:

`first` - the first map to compose

`second` - the second map to compose

Returns:

a new map that composes the given two maps

Since:

6.2

compositeMap

```
public static <K,V> Map<K,V> compositeMap(Map<K,V> first,
                                          Map<K,V> second,
                                          @Nullable
                                          BiFunction<K,V,V> putFunction,
                                          @Nullable
                                          Consumer<Map<K,V>> putAllFunction)
```

Return a map that combines the provided maps. Invoking `Map.put(Object, Object)` on the turned map will apply `putFunction`, or will throw an `UnsupportedOperationException` `tFunction` is `null`. The same applies to `Map.putAll(Map)` and `putAllFunction`.

In the case of a key collision, `first` takes precedence over `second`. In other words, entries in `second` with a key that is also mapped by `first` are effectively ignored.

Parameters:

`first` - the first map to compose

`second` - the second map to compose

`putFunction` - applied when `Map::put` is invoked. If `null`, `Map::put` throws an `UnsupportedOperationException`.

`putAllFunction` - applied when `Map::putAll` is invoked. If `null`, `Map::putAll` throws an `UnsupportedOperationException`.

Returns:

a new map that composes the give maps

Since:

6.2