Module java.base
Package java.util.concurrent

Class ConcurrentHashMap<K,V>

java.lang.Object
    java.util.AbstractMap<K,V>
        java.util.concurrent.ConcurrentHashMap<K,V>

Type Parameters:

K - the type of keys maintained by this map

V - the type of mapped values

All Implemented Interfaces:

Serializable, ConcurrentMap<K,V>, Map<K,V>

```
public class ConcurrentHashMap<K,V>
extends AbstractMap<K,V>
implements ConcurrentMap<K,V>, Serializable
```

A hash table supporting full concurrency of retrievals and high expected concurrency for updates. This class obeys the same functional specification as Hashtable, and includes versions of methods corresponding to each method of Hashtable. However, even though all operations are thread-safe, retrieval operations do *not* entail locking, and there is *not* any support for locking the entire table in a way that prevents all access. This class is fully interoperable with Hashtable in programs that rely on its thread safety but not on its synchronization details.

Retrieval operations (including get) generally do not block, so may overlap with update operations (including put and remove). Retrievals reflect the results of the most recently *completed* update operations holding upon their onset. (More formally, an update operation for a given key bears a *happens-before* relation with any (non-null) retrieval for that key reporting the updated value.) For aggregate operations such as putAll and clear, concurrent retrievals may reflect insertion or removal of only some entries. Similarly, Iterators, Spliterators and Enumerations return elements reflecting the state of the hash table at some point at or since the creation of the iterator/enumeration. They do *not* throw ConcurrentModificationException. However, iterators are designed to be used by only one thread at a time. Bear in mind that the results of aggregate status methods including size, isEmpty, and containsValue are typically useful only when a map is not undergoing concurrent updates in other threads. Otherwise the results of these methods reflect transient states that may be adequate for monitoring or estimation purposes, but not for program control.

The table is dynamically expanded when there are too many collisions (i.e., keys that have distinct hash codes but fall into the same slot modulo the table size), with the expected average effect of maintaining roughly two bins per mapping (corresponding to a 0.75 load factor threshold for resizing). There may be much variance around this average as mappings are added and removed, but overall, this maintains a commonly accepted time/space tradeoff for hash tables. However, resizing this or any other kind of hash table may be a relatively slow operation. When possible, it is a good idea to provide a size estimate as an optional initialCapacity constructor argument. An additional optional loadFactor constructor argument provides a further means of customizing initial table capacity by specifying the

table density to be used in calculating the amount of space to allocate for the given number of elements. Also, for compatibility with previous versions of this class, constructors may optionally specify an expected `concurrencyLevel` as an additional hint for internal sizing. Note that using many keys with exactly the same `hashCode()` is a sure way to slow down performance of any hash table. To ameliorate impact, when keys are `Comparable`, this class may use comparison order among keys to help break ties.

A `Set` projection of a ConcurrentHashMap may be created (using `newKeySet()` or `newKeySet(int)`), or viewed (using `keySet(Object)` when only keys are of interest, and the mapped values are (perhaps transiently) not used or all take the same mapping value.

A ConcurrentHashMap can be used as a scalable frequency map (a form of histogram or multiset) by using `LongAdder` values and initializing via `computeIfAbsent`. For example, to add a count to a `ConcurrentHashMap<String,LongAdder> freqs`, you can use `freqs.computeIfAbsent(key, k -> new LongAdder()).increment();`

This class and its views and iterators implement all of the *optional* methods of the `Map` and `Iterator` interfaces.

Like `Hashtable` but unlike `HashMap`, this class does *not* allow `null` to be used as a key or value.

ConcurrentHashMaps support a set of sequential and parallel bulk operations that, unlike most `Stream` methods, are designed to be safely, and often sensibly, applied even with maps that are being concurrently updated by other threads; for example, when computing a snapshot summary of the values in a shared registry. There are three kinds of operation, each with four forms, accepting functions with keys, values, entries, and (key, value) pairs as arguments and/or return values. Because the elements of a ConcurrentHashMap are not ordered in any particular way, and may be processed in different orders in different parallel executions, the correctness of supplied functions should not depend on any ordering, or on any other objects or values that may transiently change while computation is in progress; and except for forEach actions, should ideally be side-effect-free. Bulk operations on `Map.Entry` objects do not support method `setValue`.

- forEach: Performs a given action on each element. A variant form applies a given transformation on each element before performing the action.
- search: Returns the first available non-null result of applying a given function on each element; skipping further search when a result is found.
- reduce: Accumulates each element. The supplied reduction function cannot rely on ordering (more formally, it should be both associative and commutative). There are five variants:
    - Plain reductions. (There is not a form of this method for (key, value) function arguments since there is no corresponding return type.)
    - Mapped reductions that accumulate the results of a given function applied to each element.
    - Reductions to scalar doubles, longs, and ints, using a given basis value.

These bulk operations accept a `parallelismThreshold` argument. Methods proceed sequentially if the current map size is estimated to be less than the given threshold. Using a value of `Long.MAX_VALUE` suppresses all parallelism. Using a value of `1` results in maximal parallelism by partitioning into enough subtasks to fully utilize the `ForkJoinPool.commonPool()` that is used for all parallel computations. Normally, you would initially choose one of these extreme values, and then measure performance of using in-between values that trade off overhead versus throughput.

The concurrency properties of bulk operations follow from those of ConcurrentHashMap: Any non-null result returned from `get(key)` and related access methods bears a happens-before relation with the associated insertion or update. The result of any bulk operation reflects the composition of these per-element relations (but is not necessarily atomic with respect to the map as a whole unless it is somehow known to be quiescent). Conversely, because keys and values in the map are never null, null serves as a reliable atomic indicator of the current lack of any result. To maintain this property, null serves as an implicit basis for all non-scalar reduction operations. For the double, long, and int versions, the basis should be one that, when combined with any other value, returns that other value (more formally, it should be the identity element for the reduction). Most common reductions have these properties; for example, computing a sum with basis 0 or a minimum with basis MAX_VALUE.

Search and transformation functions provided as arguments should similarly return null to indicate the lack of any result (in which case it is not used). In the case of mapped reductions, this also enables transformations to serve as filters, returning null (or, in the case of primitive specializations, the identity basis) if the element should not be combined. You can create compound transformations and filterings by composing them yourself under this "null means there is nothing there now" rule before using them in search or reduce operations.

Methods accepting and/or returning Entry arguments maintain key-value associations. They may be useful for example when finding the key for the greatest value. Note that "plain" Entry arguments can be supplied using `new AbstractMap.SimpleEntry(k,v)`.

Bulk operations may complete abruptly, throwing an exception encountered in the application of a supplied function. Bear in mind when handling such exceptions that other concurrently executing functions could also have thrown exceptions, or would have done so if the first exception had not occurred.

Speedups for parallel compared to sequential forms are common but not guaranteed. Parallel operations involving brief functions on small maps may execute more slowly than sequential forms if the underlying work to parallelize the computation is more expensive than the computation itself. Similarly, parallelization may not lead to much actual parallelism if all processors are busy performing unrelated tasks.

All arguments to all task methods must be non-null.

This class is a member of the Java Collections Framework.

Since:

1.5

See Also:

Serialized Form

Nested Class Summary

Nested Classes

Modifier and Type	Class	Description
static class	**ConcurrentHashMap.KeySetView<K,V>**	A view of a ConcurrentHashMap as a Set of keys, in which additions may optionally be

enabled by mapping to a common value.

Nested classes/interfaces declared in class java.util.AbstractMap

AbstractMap.SimpleEntry<K,V>, AbstractMap.SimpleImmutableEntry<K,V>

Nested classes/interfaces declared in interface java.util.Map

Map.Entry<K,V>

Constructor Summary

Constructors

Constructor	Description
`ConcurrentHashMap()`	Creates a new, empty map with the default initial table size (16).
`ConcurrentHashMap (int initialCapacity)`	Creates a new, empty map with an initial table size accommodating the specified number of elements without the need to dynamically resize.
`ConcurrentHashMap (int initialCapacity, float loadFactor)`	Creates a new, empty map with an initial table size based on the given number of elements (`initialCapacity`) and initial table density (`loadFactor`).
`ConcurrentHashMap (int initialCapacity, float loadFactor, int concurrencyLevel)`	Creates a new, empty map with an initial table size based on the given number of elements (`initialCapacity`), initial table density (`loadFactor`), and number of concurrently updating threads (`concurrencyLevel`).
`ConcurrentHashMap(Map<? extends K,? extends V> m)`	Creates a new map with the same mappings as the given map.

Method Summary

All Methods Static Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
`void`	`clear()`	Removes all of the mappings from this map.

V	compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	Attempts to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
V	computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)	If the specified key is not already associated with a value, attempts to compute its value using the given mapping function and enters it into this map unless null.
V	computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	If the value for the specified key is present, attempts to compute a new mapping given the key and its current mapped value.
boolean	contains(Object value)	Tests if some key maps into the specified value in this table.
boolean	containsKey(Object key)	Tests if the specified object is a key in this table.
boolean	containsValue (Object value)	Returns true if this map maps one or more keys to the specified value.
Enumeration<V>	elements()	Returns an enumeration of the values in this table.
Set<Map.Entry<K,V>>	entrySet()	Returns a Set view of the mappings contained in this map.
boolean	equals(Object o)	Compares the specified object with this map for equality.
void	forEach (long parallelismThreshold BiConsumer<? super K,? super V> action)	Performs the given action for each (key, value).
<U> void	forEach (long parallelismThreshold BiFunction<? super K,? super V,? extends U> transformer,	Performs the given action for each non-null transformation of each (key, value).

	Consumer<? super U> action)	
void	forEachEntry(long parallelismThreshold, Consumer<? super Map.Entry<K,V>> action)	Performs the given action for each entry.
<U> void	forEachEntry(long parallelismThreshold, Function<Map.Entry<K,V>,? extends U> transformer, Consumer<? super U> action)	Performs the given action for each non-null transformation of each entry.
void	forEachKey(long parallelismThreshold, Consumer<? super K> action)	Performs the given action for each key.
<U> void	forEachKey(long parallelismThreshold, Function<? super K,? extends U> transformer, Consumer<? super U> action)	Performs the given action for each non-null transformation of each key.
void	forEachValue(long parallelismThreshold, Consumer<? super V> action)	Performs the given action for each value.
<U> void	forEachValue(long parallelismThreshold, Function<? super V,? extends U> transformer, Consumer<? super U> action)	Performs the given action for each non-null transformation of each value.
V	get(Object key)	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
V	getOrDefault(Object key, V defaultValue)	Returns the value to which the specified key is mapped, or the given default value if

		this map contains no mapping for the key.
int	**hashCode**()	Returns the hash code value for this Map, i.e., the sum of, for each key-value pair in the map, key.hashCode() ^ value.hashCode().
boolean	**isEmpty**()	Returns true if this map contains no key-value mappings.
Enumeration<**K**>	**keys**()	Returns an enumeration of the keys in this table.
ConcurrentHashMap.KeyS **V**>	**keySet**()	Returns a Set view of the keys contained in this map.
ConcurrentHashMap.KeyS **V**>	**keySet**(**V** mappedValue)	Returns a Set view of the keys in this map, using the given common mapped value for any additions (i.e., Collection.add(E) and Collection.addAll(Collection)).
long	**mappingCount**()	Returns the number of mappings.
V	**merge**(**K** key, **V** value, **BiFunction**<? super **V**,? super **V**,? extends **V**> remappingFunction)	If the specified key is not already associated with a (non-null) value, associates it with the given value.
static <K> **ConcurrentHashMap.** **Boolean**>	**newKeySet**()	Creates a new Set backed by a ConcurrentHashMap from the given type to Boolean.TRUE.
static <K> **ConcurrentHashMap.** **Boolean**>	**newKeySet**(int initialCapacity)	Creates a new Set backed by a ConcurrentHashMap from the given type to Boolean.TRUE.
V	**put**(**K** key, **V** value)	Maps the specified key to the specified value in this table.
void	**putAll**(**Map**<? extends **K**,? extends **V**> m)	Copies all of the mappings from the specified map to this one.
V	**putIfAbsent**(**K** key, **V** value)	If the specified key is not already associated with a

		value, associates it with the given value.
<U> U	**reduce** (long parallelismThreshold **BiFunction**<? super **K**,? super **V**,? extends U> transformer, **BiFunction**<? super U,? super U,? extends U> reducer)	Returns the result of accumulating the given transformation of all (key, value) pairs using the given reducer to combine values, or null if none.
Map.Entry<**K**,**V**>	**reduceEntries** (long parallelismThreshold **BiFunction**<**Map.Entry**<**K**, **V**>,**Map.Entry**<**K**,**V**>,? extends **Map.Entry**<**K**, **V**>> reducer)	Returns the result of accumulating all entries using the given reducer to combine values, or null if none.
<U> U	**reduceEntries** (long parallelismThreshold **Function**<**Map.Entry**<**K**,**V**>,? extends U> transformer, **BiFunction**<? super U,? super U,? extends U> reducer)	Returns the result of accumulating the given transformation of all entries using the given reducer to combine values, or null if none.
double	**reduceEntriesToDouble** (long parallelismThreshold **ToDoubleFunction**<**Map.Entry** **V**>> transformer, double basis, **DoubleBinaryOperator** reduc	Returns the result of accumulating the given transformation of all entries using the given reducer to combine values, and the given basis as an identity value.
int	**reduceEntriesToInt** (long parallelismThreshold **ToIntFunction**<**Map.Entry**<**K**, **V**>> transformer, int basis, **IntBinaryOperator** reducer)	Returns the result of accumulating the given transformation of all entries using the given reducer to combine values, and the given basis as an identity value.
long	**reduceEntriesToLong** (long parallelismThreshold **ToLongFunction**<**Map.Entry**<**K** **V**>> transformer, long basis, **LongBinaryOperator** reducer	Returns the result of accumulating the given transformation of all entries using the given reducer to combine values, and the given basis as an identity value.

K	**reduceKeys**(long parallelismThreshold, **BiFunction**<? super **K**,? super **K**,? extends **K**> reducer)	Returns the result of accumulating all keys using the given reducer to combine values, or null if none.
<U> U	**reduceKeys**(long parallelismThreshold, **Function**<? super **K**,? extends U> transformer, **BiFunction**<? super U,? super U,? extends U> reducer)	Returns the result of accumulating the given transformation of all keys using the given reducer to combine values, or null if none.
double	**reduceKeysToDouble**(long parallelismThreshold, **ToDoubleFunction**<? super **K**> transformer, double basis, **DoubleBinaryOperator** reduc	Returns the result of accumulating the given transformation of all keys using the given reducer to combine values, and the given basis as an identity value.
int	**reduceKeysToInt**(long parallelismThreshold, **ToIntFunction**<? super **K**> transformer, int basis, **IntBinaryOperator** reducer)	Returns the result of accumulating the given transformation of all keys using the given reducer to combine values, and the given basis as an identity value.
long	**reduceKeysToLong**(long parallelismThreshold, **ToLongFunction**<? super **K**> transformer, long basis, **LongBinaryOperator** reducer	Returns the result of accumulating the given transformation of all keys using the given reducer to combine values, and the given basis as an identity value.
double	**reduceToDouble**(long parallelismThreshold, **ToDoubleBiFunction**<? super **K**,? super **V**> transformer, double basis, **DoubleBinaryOperator** reduc	Returns the result of accumulating the given transformation of all (key, value) pairs using the given reducer to combine values, and the given basis as an identity value.
int	**reduceToInt**(long parallelismThreshold, **ToIntBiFunction**<? super **K**,? super **V**> transformer,	Returns the result of accumulating the given transformation of all (key, value) pairs using the given reducer to combine values,

	int basis, IntBinaryOperator reducer)	and the given basis as an identity value.
long	reduceToLong (long parallelismThreshold ToLongBiFunction<? super K,? super V> transformer, long basis, LongBinaryOperator reducer	Returns the result of accumulating the given transformation of all (key, value) pairs using the given reducer to combine values, and the given basis as an identity value.
V	reduceValues (long parallelismThreshold BiFunction<? super V,? super V,? extends V> reducer)	Returns the result of accumulating all values using the given reducer to combine values, or null if none.
<U> U	reduceValues (long parallelismThreshold Function<? super V,? extends U> transformer, BiFunction<? super U,? super U,? extends U> reducer)	Returns the result of accumulating the given transformation of all values using the given reducer to combine values, or null if none.
double	reduceValuesToDouble (long parallelismThreshold ToDoubleFunction<? super V> transformer, double basis, DoubleBinaryOperator reduc	Returns the result of accumulating the given transformation of all values using the given reducer to combine values, and the given basis as an identity value.
int	reduceValuesToInt (long parallelismThreshold ToIntFunction<? super V> transformer, int basis, IntBinaryOperator reducer)	Returns the result of accumulating the given transformation of all values using the given reducer to combine values, and the given basis as an identity value.
long	reduceValuesToLong (long parallelismThreshold ToLongFunction<? super V> transformer, long basis, LongBinaryOperator reducer	Returns the result of accumulating the given transformation of all values using the given reducer to combine values, and the given basis as an identity value.
V	remove(Object key)	Removes the key (and its corresponding value) from

		this map.
boolean	**remove**(**Object** key, **Object** value)	Removes the entry for a key only if currently mapped to a given value.
V	**replace**(**K** key, **V** value)	Replaces the entry for a key only if currently mapped to some value.
boolean	**replace**(**K** key, **V** oldValue, **V** newValue)	Replaces the entry for a key only if currently mapped to a given value.
<U> U	**search** (long parallelismThreshold **BiFunction**<? super **K**,? super **V**,? extends U> searchFunction)	Returns a non-null result from applying the given search function on each (key, value), or null if none.
<U> U	**searchEntries** (long parallelismThreshold **Function**<**Map.Entry**<**K**,**V**>,? extends U> searchFunction)	Returns a non-null result from applying the given search function on each entry, or null if none.
<U> U	**searchKeys** (long parallelismThreshold **Function**<? super **K**,? extends U> searchFunction)	Returns a non-null result from applying the given search function on each key, or null if none.
<U> U	**searchValues** (long parallelismThreshold **Function**<? super **V**,? extends U> searchFunction)	Returns a non-null result from applying the given search function on each value, or null if none.
int	**size**()	Returns the number of key-value mappings in this map.
String	**toString**()	Returns a string representation of this map.
Collection<**V**>	**values**()	Returns a `Collection` view of the values contained in this map.

Methods declared in class java.util.AbstractMap

clone, isEmpty, size

Methods declared in class java.lang.Object

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

Methods declared in interface java.util.concurrent.ConcurrentMap

forEach, replaceAll

Constructor Details

ConcurrentHashMap

```
public ConcurrentHashMap()
```

Creates a new, empty map with the default initial table size (16).

ConcurrentHashMap

```
public ConcurrentHashMap(int initialCapacity)
```

Creates a new, empty map with an initial table size accommodating the specified number of elements without the need to dynamically resize.

Parameters:

`initialCapacity` - The implementation performs internal sizing to accommodate this many elements.

Throws:

`IllegalArgumentException` - if the initial capacity of elements is negative

ConcurrentHashMap

```
public ConcurrentHashMap(Map<? extends K,? extends V> m)
```

Creates a new map with the same mappings as the given map.

Parameters:

`m` - the map

ConcurrentHashMap

```
public ConcurrentHashMap(int initialCapacity,
                         float loadFactor)
```

Creates a new, empty map with an initial table size based on the given number of elements (`initialCapacity`) and initial table density (`loadFactor`).

Parameters:

`initialCapacity` - the initial capacity. The implementation performs internal sizing to accommodate this many elements, given the specified load factor.

`loadFactor` - the load factor (table density) for establishing the initial table size

Throws:

`IllegalArgumentException` - if the initial capacity of elements is negative or the load factor is nonpositive

Since:

1.6

ConcurrentHashMap

```
public ConcurrentHashMap(int initialCapacity,
                         float loadFactor,
                         int concurrencyLevel)
```

Creates a new, empty map with an initial table size based on the given number of elements (`initialCapacity`), initial table density (`loadFactor`), and number of concurrently updating threads (`concurrencyLevel`).

Parameters:

`initialCapacity` - the initial capacity. The implementation performs internal sizing to accommodate this many elements, given the specified load factor.

`loadFactor` - the load factor (table density) for establishing the initial table size

`concurrencyLevel` - the estimated number of concurrently updating threads. The implementation may use this value as a sizing hint.

Throws:

`IllegalArgumentException` - if the initial capacity is negative or the load factor or concurrencyLevel are nonpositive

Method Details

size

```
public int size()
```

Returns the number of key-value mappings in this map. If the map contains more than `Integer.MAX_VALUE` elements, returns `Integer.MAX_VALUE`.

Specified by:

`size` in interface `Map<K,V>`

Overrides:

`size` in class `AbstractMap<K,V>`

Returns:

the number of key-value mappings in this map

isEmpty

```
public boolean isEmpty()
```

Returns `true` if this map contains no key-value mappings.

Specified by:

`isEmpty` in interface `Map<K,V>`

Overrides:

`isEmpty` in class `AbstractMap<K,V>`

Returns:

`true` if this map contains no key-value mappings

get

```
public V get(Object key)
```

Returns the value to which the specified key is mapped, or `null` if this map contains no mapping for the key.

More formally, if this map contains a mapping from a key `k` to a value `v` such that `key.equals(k)`, then this method returns `v`; otherwise it returns `null`. (There can be at most one such mapping.)

Specified by:

`get` in interface `Map<K,V>`

Overrides:

`get` in class `AbstractMap<K,V>`

Parameters:

`key` - the key whose associated value is to be returned

Returns:

the value to which the specified key is mapped, or `null` if this map contains no mapping for the key

Throws:

`NullPointerException` - if the specified key is null

containsKey

```
public boolean containsKey(Object key)
```

Tests if the specified object is a key in this table.

Specified by:

`containsKey` in interface `Map<K,V>`

Overrides:

`containsKey` in class `AbstractMap<K,V>`

Parameters:

`key` - possible key

Returns:

`true` if and only if the specified object is a key in this table, as determined by the `equals` method; `false` otherwise

Throws:

`NullPointerException` - if the specified key is null

containsValue

```
public boolean containsValue(Object value)
```

Returns `true` if this map maps one or more keys to the specified value. Note: This method may require a full traversal of the map, and is much slower than method `containsKey`.

Specified by:

`containsValue` in interface `Map<K,V>`

Overrides:

`containsValue` in class `AbstractMap<K,V>`

Parameters:

`value` - value whose presence in this map is to be tested

Returns:

`true` if this map maps one or more keys to the specified value

Throws:

`NullPointerException` - if the specified value is null

put

```
public V put(K key,
             V value)
```

Maps the specified key to the specified value in this table. Neither the key nor the value can be null.

The value can be retrieved by calling the `get` method with a key that is equal to the original key.

Specified by:

`put` in interface `Map<K,V>`

Overrides:

`put` in class `AbstractMap<K,V>`

Parameters:

`key` - key with which the specified value is to be associated

`value` - value to be associated with the specified key

Returns:

the previous value associated with `key`, or `null` if there was no mapping for `key`

Throws:

`NullPointerException` - if the specified key or value is null

putAll

```
public void putAll(Map<? extends K,? extends V> m)
```

Copies all of the mappings from the specified map to this one. These mappings replace any mappings that this map had for any of the keys currently in the specified map.

Specified by:

`putAll` in interface `Map<K,V>`

Overrides:

`putAll` in class `AbstractMap<K,V>`

Parameters:

`m` - mappings to be stored in this map

remove

```
public V remove(Object key)
```

Removes the key (and its corresponding value) from this map. This method does nothing if the key is not in the map.

Specified by:

`remove` in interface `Map<K,V>`

Overrides:

`remove` in class `AbstractMap<K,V>`

Parameters:

`key` - the key that needs to be removed

Returns:

the previous value associated with `key`, or `null` if there was no mapping for `key`

Throws:

`NullPointerException` - if the specified key is null

clear

```
public void clear()
```

Removes all of the mappings from this map.

Specified by:

clear in interface Map<K,V>

Overrides:

clear in class AbstractMap<K,V>

keySet

```
public ConcurrentHashMap.KeySetView<K,V> keySet()
```

Returns a Set view of the keys contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. The set supports element removal, which removes the corresponding mapping from this map, via the Iterator.remove, Set.remove, removeAll, retainAll, and clear operations. It does not support the add or addAll operations.

The view's iterators and spliterators are *weakly consistent*.

The view's spliterator reports Spliterator.CONCURRENT, Spliterator.DISTINCT, and Spliterator.NONNULL.

Specified by:

keySet in interface Map<K,V>

Overrides:

keySet in class AbstractMap<K,V>

Returns:

the set view

values

```
public Collection<V> values()
```

Returns a Collection view of the values contained in this map. The collection is backed by the map, so changes to the map are reflected in the collection, and vice-versa. The collection supports element removal, which removes the corresponding mapping from this map, via the Iterator.remove, Collection.remove, removeAll, retainAll, and clear operations. It does not support the add or addAll operations.

The view's iterators and spliterators are *weakly consistent*.

The view's spliterator reports Spliterator.CONCURRENT and Spliterator.NONNULL.

Specified by:

values in interface Map<K,V>

Overrides:

values in class AbstractMap<K,V>

Returns:

the collection view

entrySet

```
public Set<Map.Entry<K,V>> entrySet()
```

Returns a `Set` view of the mappings contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. The set supports element removal, which removes the corresponding mapping from the map, via the `Iterator.remove`, `Set.remove`, `removeAll`, `retainAll`, and `clear` operations.

The view's iterators and spliterators are *weakly consistent*.

The view's `spliterator` reports `Spliterator.CONCURRENT`, `Spliterator.DISTINCT`, and `Spliterator.NONNULL`.

Specified by:

`entrySet` in interface `Map<K,V>`

Returns:

the set view

hashCode

```
public int hashCode()
```

Returns the hash code value for this `Map`, i.e., the sum of, for each key-value pair in the map, `key.hashCode() ^ value.hashCode()`.

Specified by:

`hashCode` in interface `Map<K,V>`

Overrides:

`hashCode` in class `AbstractMap<K,V>`

Returns:

the hash code value for this map

See Also:

`Map.Entry.hashCode()`, `Object.equals(Object)`, `Set.equals(Object)`

toString

```
public String toString()
```

Returns a string representation of this map. The string representation consists of a list of key-value mappings (in no particular order) enclosed in braces ("{}"). Adjacent mappings are separated by the characters `", "` (comma and space). Each key-value mapping is rendered as the key followed by an equals sign ("=") followed by the associated value.

Overrides:

`toString` in class `AbstractMap<K,V>`

Returns:

a string representation of this map

equals

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

Compares the specified object with this map for equality. Returns `true` if the given object is a map with the same mappings as this map. This operation may return misleading results if either map is concurrently modified during execution of this method.

Specified by:

`equals` in interface `Map<K,V>`

Overrides:

`equals` in class `AbstractMap<K,V>`

Parameters:

`o` - object to be compared for equality with this map

Returns:

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

See Also:

`Object.hashCode()`, `HashMap`

putIfAbsent

```
public V putIfAbsent(K key,
                     V value)
```

If the specified key is not already associated with a value, associates it with the given value. This is equivalent to, for this `map`:

```
 if (!map.containsKey(key))
   return map.put(key, value);
 else
   return map.get(key);
```

except that the action is performed atomically.

Specified by:

`putIfAbsent` in interface `ConcurrentMap<K,V>`

Specified by:

`putIfAbsent` in interface `Map<K,V>`

Parameters:

`key` - key with which the specified value is to be associated

`value` - value to be associated with the specified key

Returns:

the previous value associated with the specified key, or `null` if there was no mapping for the key

Throws:

`NullPointerException` - if the specified key or value is null

remove

```
public boolean remove(Object key,
                      Object value)
```

Removes the entry for a key only if currently mapped to a given value. This is equivalent to, for this `map`:

```
if (map.containsKey(key)
    && Objects.equals(map.get(key), value)) {
  map.remove(key);
  return true;
} else {
  return false;
}
```

except that the action is performed atomically.

Specified by:

`remove` in interface `ConcurrentMap<K,V>`

Specified by:

`remove` in interface `Map<K,V>`

Parameters:

`key` - key with which the specified value is associated

`value` - value expected to be associated with the specified key

Returns:

`true` if the value was removed

Throws:

`NullPointerException` - if the specified key is null

replace

```
public boolean replace(K key,
                       V oldValue,
                       V newValue)
```

Replaces the entry for a key only if currently mapped to a given value. This is equivalent to, for this `map`:

```
if (map.containsKey(key)
    && Objects.equals(map.get(key), oldValue)) {
```

```
  map.put(key, newValue);
  return true;
} else {
  return false;
}
```

except that the action is performed atomically.

Specified by:

`replace` in interface `ConcurrentMap<K,V>`

Specified by:

`replace` in interface `Map<K,V>`

Parameters:

`key` - key with which the specified value is associated

`oldValue` - value expected to be associated with the specified key

`newValue` - value to be associated with the specified key

Returns:

`true` if the value was replaced

Throws:

`NullPointerException` - if any of the arguments are null

replace

```
public V replace(K key,
                 V value)
```

Replaces the entry for a key only if currently mapped to some value. This is equivalent to, for this `map`:

```
if (map.containsKey(key))
  return map.put(key, value);
else
  return null;
```

except that the action is performed atomically.

Specified by:

`replace` in interface `ConcurrentMap<K,V>`

Specified by:

`replace` in interface `Map<K,V>`

Parameters:

`key` - key with which the specified value is associated

`value` - value to be associated with the specified key

Returns:

the previous value associated with the specified key, or `null` if there was no mapping for the key

Throws:

`NullPointerException` - if the specified key or value is null

getOrDefault

```
public V getOrDefault(Object key,
                      V defaultValue)
```

Returns the value to which the specified key is mapped, or the given default value if this map contains no mapping for the key.

Specified by:

`getOrDefault` in interface `ConcurrentMap<K,V>`

Specified by:

`getOrDefault` in interface `Map<K,V>`

Parameters:

`key` - the key whose associated value is to be returned

`defaultValue` - the value to return if this map contains no mapping for the given key

Returns:

the mapping for the key, if present; else the default value

Throws:

`NullPointerException` - if the specified key is null

computeIfAbsent

```
public V computeIfAbsent(K key,
                         Function<? super K,? extends V> mappingFunction)
```

If the specified key is not already associated with a value, attempts to compute its value using the given mapping function and enters it into this map unless `null`. The entire method invocation is performed atomically. The supplied function is invoked exactly once per invocation of this method if the key is absent, else not at all. Some attempted update operations on this map by other threads may be blocked while computation is in progress, so the computation should be short and simple.

The mapping function must not modify this map during computation.

Specified by:

`computeIfAbsent` in interface `ConcurrentMap<K,V>`

Specified by:

`computeIfAbsent` in interface `Map<K,V>`

Parameters:

`key` - key with which the specified value is to be associated

`mappingFunction` - the function to compute a value

Returns:

the current (existing or computed) value associated with the specified key, or null if the computed value is null

Throws:

`NullPointerException` - if the specified key or mappingFunction is null

`IllegalStateException` - if the computation detectably attempts a recursive update to this map that would otherwise never complete

`RuntimeException` - or Error if the mappingFunction does so, in which case the mapping is left unestablished

computeIfPresent

```
public V computeIfPresent
(K key,
 BiFunction<? super K,? super V,? extends V> remappingFunction)
```

If the value for the specified key is present, attempts to compute a new mapping given the key and its current mapped value. The entire method invocation is performed atomically. The supplied function is invoked exactly once per invocation of this method if the key is present, else not at all. Some attempted update operations on this map by other threads may be blocked while computation is in progress, so the computation should be short and simple.

The remapping function must not modify this map during computation.

Specified by:

`computeIfPresent` in interface `ConcurrentMap<K,V>`

Specified by:

`computeIfPresent` in interface `Map<K,V>`

Parameters:

`key` - key with which a value may be associated

`remappingFunction` - the function to compute a value

Returns:

the new value associated with the specified key, or null if none

Throws:

`NullPointerException` - if the specified key or remappingFunction is null

`IllegalStateException` - if the computation detectably attempts a recursive update to this map that would otherwise never complete

`RuntimeException` - or Error if the remappingFunction does so, in which case the mapping is unchanged

compute

```
public V compute
(K key,
 BiFunction<? super K,? super V,? extends V> remappingFunction)
```

Attempts to compute a mapping for the specified key and its current mapped value (or `null` if there is no current mapping). The entire method invocation is performed atomically. The supplied function is invoked exactly once per invocation of this method. Some attempted update operations on this map by other threads may be blocked while computation is in progress, so the computation should be short and simple.

The remapping function must not modify this map during computation.

Specified by:

`compute` in interface `ConcurrentMap<K,V>`

Specified by:

`compute` in interface `Map<K,V>`

Parameters:

`key` - key with which the specified value is to be associated

`remappingFunction` - the function to compute a value

Returns:

the new value associated with the specified key, or null if none

Throws:

`NullPointerException` - if the specified key or remappingFunction is null

`IllegalStateException` - if the computation detectably attempts a recursive update to this map that would otherwise never complete

`RuntimeException` - or Error if the remappingFunction does so, in which case the mapping is unchanged

merge

```
public V merge
(K key,
 V value,
 BiFunction<? super V,? super V,? extends V> remappingFunction)
```

If the specified key is not already associated with a (non-null) value, associates it with the given value. Otherwise, replaces the value with the results of the given remapping function, or removes if `null`. The entire method invocation is performed atomically. Some attempted update operations on this map by other threads may be blocked while computation is in progress, so the computation should be short and simple, and must not attempt to update any other mappings of this Map.

Specified by:

`merge` in interface `ConcurrentMap<K,V>`

Specified by:

`merge` in interface `Map<K,V>`

Parameters:

`key` - key with which the specified value is to be associated

`value` - the value to use if absent

`remappingFunction` - the function to recompute a value if present

Returns:

the new value associated with the specified key, or null if none

Throws:

`NullPointerException` - if the specified key or the remappingFunction is null

`RuntimeException` - or Error if the remappingFunction does so, in which case the mapping is unchanged

contains

```
public boolean contains(Object value)
```

Tests if some key maps into the specified value in this table.

Note that this method is identical in functionality to `containsValue(Object)`, and exists solely to ensure full compatibility with class `Hashtable`, which supported this method prior to introduction of the Java Collections Framework.

Parameters:

`value` - a value to search for

Returns:

`true` if and only if some key maps to the `value` argument in this table as determined by the `equals` method; `false` otherwise

Throws:

`NullPointerException` - if the specified value is null

keys

```
public Enumeration<K> keys()
```

Returns an enumeration of the keys in this table.

Returns:

an enumeration of the keys in this table

See Also:

`keySet()`

elements

```
public Enumeration<V> elements()
```

Returns an enumeration of the values in this table.

Returns:

an enumeration of the values in this table

See Also:

`values()`

mappingCount

```
public long mappingCount()
```

Returns the number of mappings. This method should be used instead of `size()` because a ConcurrentHashMap may contain more mappings than can be represented as an int. The value returned is an estimate; the actual count may differ if there are concurrent insertions or removals.

Returns:

the number of mappings

Since:

1.8

newKeySet

```
public static <K>
ConcurrentHashMap.KeySetView<K,Boolean> newKeySet()
```

Creates a new `Set` backed by a ConcurrentHashMap from the given type to `Boolean.TRUE`.

Type Parameters:

K - the element type of the returned set

Returns:

the new set

Since:

1.8

newKeySet

```
public static <K>
ConcurrentHashMap.KeySetView<K,Boolean> newKeySet(int initialCapacity)
```

Creates a new `Set` backed by a ConcurrentHashMap from the given type to `Boolean.TRUE`.

Type Parameters:

K - the element type of the returned set

Parameters:

`initialCapacity` - The implementation performs internal sizing to accommodate this many elements.

Returns:

the new set

Throws:

`IllegalArgumentException` - if the initial capacity of elements is negative

Since:

1.8

keySet

```
public ConcurrentHashMap.KeySetView<K,V> keySet(V mappedValue)
```

Returns a `Set` view of the keys in this map, using the given common mapped value for any additions (i.e., `Collection.add(E)` and `Collection.addAll(Collection)`). This is of course only appropriate if it is acceptable to use the same value for all additions from this view.

Parameters:

`mappedValue` - the mapped value to use for any additions

Returns:

the set view

Throws:

`NullPointerException` - if the mappedValue is null

forEach

```
public void forEach(long parallelismThreshold,
                    BiConsumer<? super K,? super V> action)
```

Performs the given action for each (key, value).

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`action` - the action

Since:

1.8

forEach

```
public <U> void forEach
(long parallelismThreshold,
 BiFunction<? super K,? super V,? extends U> transformer,
 Consumer<? super U> action)
```

Performs the given action for each non-null transformation of each (key, value).

Type Parameters:

`U` - the return type of the transformer

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element, or null if there is no transformation (in which case the action is not applied)

`action` - the action

Since:

1.8

search

```
public <U> U search
(long parallelismThreshold,
 BiFunction<? super K,? super V,? extends U> searchFunction)
```

Returns a non-null result from applying the given search function on each (key, value), or null if none. Upon success, further element processing is suppressed and the results of any other parallel invocations of the search function are ignored.

Type Parameters:

`U` - the return type of the search function

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`searchFunction` - a function returning a non-null result on success, else null

Returns:

a non-null result from applying the given search function on each (key, value), or null if none

Since:

1.8

reduce

```
public <U> U reduce
(long parallelismThreshold,
 BiFunction<? super K,? super V,? extends U> transformer,
 BiFunction<? super U,? super U,? extends U> reducer)
```

Returns the result of accumulating the given transformation of all (key, value) pairs using the given reducer to combine values, or null if none.

Type Parameters:

`U` - the return type of the transformer

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element, or null if there is no transformation (in which case it is not combined)

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all (key, value) pairs

Since:

1.8

reduceToDouble

```
public double reduceToDouble
(long parallelismThreshold,
 ToDoubleBiFunction<? super K,? super V> transformer,
 double basis,
 DoubleBinaryOperator reducer)
```

Returns the result of accumulating the given transformation of all (key, value) pairs using the given reducer to combine values, and the given basis as an identity value.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element

`basis` - the identity (initial default value) for the reduction

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all (key, value) pairs

Since:

1.8

reduceToLong

```
public long reduceToLong(long parallelismThreshold,
                         ToLongBiFunction<? super K,? super V> transformer,
                         long basis,
                         LongBinaryOperator reducer)
```

Returns the result of accumulating the given transformation of all (key, value) pairs using the given reducer to combine values, and the given basis as an identity value.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element

`basis` - the identity (initial default value) for the reduction

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all (key, value) pairs

Since:

1.8

reduceToInt

```
public int reduceToInt(long parallelismThreshold,
                       ToIntBiFunction<? super K,? super V> transformer,
                       int basis,
                       IntBinaryOperator reducer)
```

Returns the result of accumulating the given transformation of all (key, value) pairs using the given reducer to combine values, and the given basis as an identity value.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element

`basis` - the identity (initial default value) for the reduction

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all (key, value) pairs

Since:

1.8

forEachKey

```
public void forEachKey(long parallelismThreshold,
                       Consumer<? super K> action)
```

Performs the given action for each key.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`action` - the action

Since:

1.8

forEachKey

```
public <U> void forEachKey(long parallelismThreshold,
                           Function<? super K,? extends U> transformer,
                           Consumer<? super U> action)
```

Performs the given action for each non-null transformation of each key.

Type Parameters:

`U` - the return type of the transformer

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element, or null if there is no transformation (in which case the action is not applied)

`action` - the action

Since:

1.8

searchKeys

```
public <U> U searchKeys(long parallelismThreshold,
                        Function<? super K,? extends U> searchFunction)
```

Returns a non-null result from applying the given search function on each key, or null if none. Upon success, further element processing is suppressed and the results of any other parallel invocations of the search function are ignored.

Type Parameters:

`U` - the return type of the search function

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`searchFunction` - a function returning a non-null result on success, else null

Returns:

a non-null result from applying the given search function on each key, or null if none

Since:

1.8

reduceKeys

```
public K reduceKeys(long parallelismThreshold,
                    BiFunction<? super K,? super K,? extends K> reducer)
```

Returns the result of accumulating all keys using the given reducer to combine values, or null if none.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`reducer` - a commutative associative combining function

Returns:

the result of accumulating all keys using the given reducer to combine values, or null if none

Since:

1.8

reduceKeys

```
public <U> U reduceKeys
(long parallelismThreshold,
 Function<? super K,? extends U> transformer,
 BiFunction<? super U,? super U,? extends U> reducer)
```

Returns the result of accumulating the given transformation of all keys using the given reducer to combine values, or null if none.

Type Parameters:

`U` - the return type of the transformer

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element, or null if there is no transformation (in which case it is not combined)

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all keys

Since:

1.8

reduceKeysToDouble

```
public double reduceKeysToDouble(long parallelismThreshold,
                                 ToDoubleFunction<? super K> transformer,
                                 double basis,
                                 DoubleBinaryOperator reducer)
```

Returns the result of accumulating the given transformation of all keys using the given reducer to combine values, and the given basis as an identity value.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element

`basis` - the identity (initial default value) for the reduction

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all keys

Since:

1.8

reduceKeysToLong

```
public long reduceKeysToLong(long parallelismThreshold,
                             ToLongFunction<? super K> transformer,
                             long basis,
                             LongBinaryOperator reducer)
```

Returns the result of accumulating the given transformation of all keys using the given reducer to combine values, and the given basis as an identity value.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element

`basis` - the identity (initial default value) for the reduction

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all keys

Since:

1.8

reduceKeysToInt

```
public int reduceKeysToInt(long parallelismThreshold,
                           ToIntFunction<? super K> transformer,
                           int basis,
                           IntBinaryOperator reducer)
```

Returns the result of accumulating the given transformation of all keys using the given reducer to combine values, and the given basis as an identity value.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element

`basis` - the identity (initial default value) for the reduction

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all keys

Since:

1.8

forEachValue

```
public void forEachValue(long parallelismThreshold,
                         Consumer<? super V> action)
```

Performs the given action for each value.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`action` - the action

Since:

1.8

forEachValue

```
public <U> void forEachValue(long parallelismThreshold,
                             Function<? super V,? extends U> transformer,
                             Consumer<? super U> action)
```

Performs the given action for each non-null transformation of each value.

Type Parameters:

`U` - the return type of the transformer

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element, or null if there is no transformation (in which case the action is not applied)

`action` - the action

Since:

1.8

searchValues

```
public <U> U searchValues(long parallelismThreshold,
                          Function<? super V,? extends U> searchFunction)
```

Returns a non-null result from applying the given search function on each value, or null if none. Upon success, further element processing is suppressed and the results of any other parallel invocations of the search function are ignored.

Type Parameters:

`U` - the return type of the search function

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`searchFunction` - a function returning a non-null result on success, else null

Returns:

a non-null result from applying the given search function on each value, or null if none

Since:

1.8

reduceValues

```
public V reduceValues(long parallelismThreshold,
                      BiFunction<? super V,? super V,? extends V> reducer)
```

Returns the result of accumulating all values using the given reducer to combine values, or null if none.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`reducer` - a commutative associative combining function

Returns:

the result of accumulating all values

Since:

1.8

reduceValues

```
public <U> U reduceValues
(long parallelismThreshold,
 Function<? super V,? extends U> transformer,
 BiFunction<? super U,? super U,? extends U> reducer)
```

Returns the result of accumulating the given transformation of all values using the given reducer to combine values, or null if none.

Type Parameters:

`U` - the return type of the transformer

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element, or null if there is no transformation (in which case it is not combined)

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all values

Since:

1.8

reduceValuesToDouble

```
public double reduceValuesToDouble(long parallelismThreshold,
                                   ToDoubleFunction<? super V> transformer,
                                   double basis,
                                   DoubleBinaryOperator reducer)
```

Returns the result of accumulating the given transformation of all values using the given reducer to combine values, and the given basis as an identity value.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element

`basis` - the identity (initial default value) for the reduction

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all values

Since:

1.8

reduceValuesToLong

```
public long reduceValuesToLong(long parallelismThreshold,
                               ToLongFunction<? super V> transformer,
                               long basis,
                               LongBinaryOperator reducer)
```

Returns the result of accumulating the given transformation of all values using the given reducer to combine values, and the given basis as an identity value.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element

`basis` - the identity (initial default value) for the reduction

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all values

Since:

1.8

reduceValuesToInt

```
public int reduceValuesToInt(long parallelismThreshold,
                             ToIntFunction<? super V> transformer,
                             int basis,
                             IntBinaryOperator reducer)
```

Returns the result of accumulating the given transformation of all values using the given reducer to combine values, and the given basis as an identity value.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element

`basis` - the identity (initial default value) for the reduction

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all values

Since:

1.8

forEachEntry

```
public void forEachEntry(long parallelismThreshold,
                         Consumer<? super Map.Entry<K,V>> action)
```

Performs the given action for each entry.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`action` - the action

Since:

1.8

forEachEntry

```
public <U> void forEachEntry
(long parallelismThreshold,
 Function<Map.Entry<K,V>,? extends U> transformer,
 Consumer<? super U> action)
```

Performs the given action for each non-null transformation of each entry.

Type Parameters:

`U` - the return type of the transformer

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element, or null if there is no transformation (in which case the action is not applied)

`action` - the action

Since:

1.8

searchEntries

```
public <U> U searchEntries
(long parallelismThreshold,
 Function<Map.Entry<K,V>,? extends U> searchFunction)
```

Returns a non-null result from applying the given search function on each entry, or null if none. Upon success, further element processing is suppressed and the results of any other parallel invocations of the search function are ignored.

Type Parameters:

`U` - the return type of the search function

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`searchFunction` - a function returning a non-null result on success, else null

Returns:

a non-null result from applying the given search function on each entry, or null if none

Since:

1.8

reduceEntries

```
public Map.Entry<K,V> reduceEntries
(long parallelismThreshold,
 BiFunction<Map.Entry<K,V>,Map.Entry<K,V>,? extends Map.Entry<K,
V>> reducer)
```

Returns the result of accumulating all entries using the given reducer to combine values, or null if none.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`reducer` - a commutative associative combining function

Returns:

the result of accumulating all entries

Since:

1.8

reduceEntries

```
public <U> U reduceEntries
(long parallelismThreshold,
 Function<Map.Entry<K,V>,? extends U> transformer,
 BiFunction<? super U,? super U,? extends U> reducer)
```

Returns the result of accumulating the given transformation of all entries using the given reducer to combine values, or null if none.

Type Parameters:

`U` - the return type of the transformer

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element, or null if there is no transformation (in which case it is not combined)

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all entries

Since:

1.8

reduceEntriesToDouble

```
public double reduceEntriesToDouble
(long parallelismThreshold,
 ToDoubleFunction<Map.Entry<K,V>> transformer,
 double basis,
 DoubleBinaryOperator reducer)
```

Returns the result of accumulating the given transformation of all entries using the given reducer to combine values, and the given basis as an identity value.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element

`basis` - the identity (initial default value) for the reduction

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all entries

Since:

1.8

reduceEntriesToLong

```
public long reduceEntriesToLong(long parallelismThreshold,
                                ToLongFunction<Map.Entry<K,V>> transformer,
                                long basis,
                                LongBinaryOperator reducer)
```

Returns the result of accumulating the given transformation of all entries using the given reducer to combine values, and the given basis as an identity value.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element

`basis` - the identity (initial default value) for the reduction

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all entries

Since:

1.8

reduceEntriesToInt

```
public int reduceEntriesToInt(long parallelismThreshold,
                              ToIntFunction<Map.Entry<K,V>> transformer,
                              int basis,
                              IntBinaryOperator reducer)
```

Returns the result of accumulating the given transformation of all entries using the given reducer to combine values, and the given basis as an identity value.

Parameters:

`parallelismThreshold` - the (estimated) number of elements needed for this operation to be executed in parallel

`transformer` - a function returning the transformation for an element

`basis` - the identity (initial default value) for the reduction

`reducer` - a commutative associative combining function

Returns:

the result of accumulating the given transformation of all entries

Since:

1.8
