Module java.base
Package java.util.stream

Interface BaseStream<T,S extends BaseStream<T,S>>

Type Parameters:
T - the type of the stream elements

S - the type of the stream implementing `BaseStream`

All Superinterfaces:
`AutoCloseable`

All Known Subinterfaces:
`DoubleStream`, `IntStream`, `LongStream`, `Stream<T>`

```
public interface BaseStream<T,S extends BaseStream<T,S>>
extends AutoCloseable
```

Base interface for streams, which are sequences of elements supporting sequential and parallel aggregate operations. The following example illustrates an aggregate operation using the stream types `Stream` and `IntStream`, computing the sum of the weights of the red widgets:

```
int sum = widgets.stream()
                 .filter(w -> w.getColor() == RED)
                 .mapToInt(w -> w.getWeight())
                 .sum();
```

See the class documentation for `Stream` and the package documentation for java.util.stream for additional specification of streams, stream operations, stream pipelines, and parallelism, which governs the behavior of all stream types.

Since:
1.8

See Also:
`Stream`, `IntStream`, `LongStream`, `DoubleStream`, java.util.stream

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method	Description
`void`	`close()`	Closes this stream, causing all close handlers for this stream pipeline to be called.
`boolean`	`isParallel()`	Returns whether this stream, if a terminal operation were to be executed, would execute in parallel.
`Iterator<T>`	`iterator()`	Returns an iterator for the elements of this stream.
`S`	`onClose(Runnable closeHandler)`	Returns an equivalent stream with an additional close handler.
`S`	`parallel()`	Returns an equivalent stream that is parallel.
`S`	`sequential()`	Returns an equivalent stream that is sequential.
`Spliterator<T>`	`spliterator()`	Returns a spliterator for the elements of this stream.
`S`	`unordered()`	Returns an equivalent stream that is unordered.

Method Details

iterator

`Iterator<T> iterator()`

Returns an iterator for the elements of this stream.

This is a terminal operation.

API Note:
This operation is provided as an "escape hatch" to enable arbitrary client-controlled pipeline traversals in the event that the existing operations are not sufficient to the task.

Returns:
the element iterator for this stream

spliterator

`Spliterator<T> spliterator()`

Returns a spliterator for the elements of this stream.

This is a terminal operation.

API Note:
This operation is provided as an "escape hatch" to enable arbitrary client-controlled pipeline traversals in the event that the existing operations are not sufficient to the task.

The returned spliterator should report the set of characteristics derived from the stream pipeline (namely the characteristics derived from the stream source spliterator and the intermediate operations). Implementations may report a sub-set of those characteristics. For example, it may be too expensive to compute the entire set for some or all possible stream pipelines.

Returns:
the element spliterator for this stream

isParallel

`boolean isParallel()`

Returns whether this stream, if a terminal operation were to be executed, would execute in parallel. Calling this method after invoking an terminal stream operation method may yield unpredictable results.

Returns:
`true` if this stream would execute in parallel if executed

sequential

`S sequential()`

Returns an equivalent stream that is sequential. May return itself, either because the stream was already sequential, or because the underlying stream state was modified to be sequential.

This is an intermediate operation.

Returns:
a sequential stream

parallel

`S parallel()`

Returns an equivalent stream that is parallel. May return itself, either because the stream was already parallel, or because the underlying stream state was modified to be parallel.

This is an intermediate operation.

Returns:
a parallel stream

unordered

`S unordered()`

Returns an equivalent stream that is unordered. May return itself, either because the stream was already unordered, or because the underlying stream state was modified to be unordered.

This is an intermediate operation.

Returns:
an unordered stream

onClose

`S onClose(Runnable closeHandler)`

Returns an equivalent stream with an additional close handler. Close handlers are run when the `close()` method is called on the stream, and are executed in the order they were added. All close handlers are run, even if earlier close handlers throw exceptions. If any close handler throws an exception, the first exception thrown will be relayed to the caller of `close()`, with any remaining exceptions added to that exception as suppressed exceptions (unless one of the remaining exceptions is the same exception as the first exception, since an exception cannot suppress itself.) May return itself.

This is an intermediate operation.

Parameters:
`closeHandler` - A task to execute when the stream is closed

Returns:
a stream with a handler that is run if the stream is closed

close

`void close()`

Closes this stream, causing all close handlers for this stream pipeline to be called.

Specified by:
`close` in interface `AutoCloseable`

See Also:
`AutoCloseable.close()`