Streams

Flags

La sorgente di uno Stream può dichiarare alcune caratteristiche che gli operatori intermedi possono verificare e che l'operazione terminale usa per prendere decisioni sulla esecuzione.

L'operazione terminale ha piena visibilità, prima di cominciare a lavorare, di quali sono le caratteristiche della pipeline di esecuzione, e può quindi prendere decisioni in merito.

Flag	Caratteristica
CONCURRENT	Parallelizzabile
DISTINCT	Elementi distinti (equals())
IMMUTABLE	Immutabile durante il consumo
NONNULL	Elementi non nulli
ORDERED	Elementi ordinati (Comparator)
SIZED	Dimensione nota
SORTED	Ordinamento definito
SUBSIZED	Suddivisioni di dimensione nota

Operazioni

```
/**
* 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.
*/
S parallel();

/**
* 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.
*/
S sequential();

/**
* Returns whether any elements of this stream match
* the provided predicate. May not evaluate the
* predicate on all elements if not necessary for
* determining the result.
*/
boolean anyMatch(Predicate< ? super T > predicate);
```

```
/**
* Returns an Optional describing some element of the
* stream, or an empty Optional if the stream is empty.
* The behavior of this operation is explicitly
* nondeterministic; it is free to select any element
* in the stream.
*/
Optional< T > findAny();

/**
* Returns a stream consisting of the elements of this
* stream, truncated to be no longer than maxSize in
* length.
*/
Stream< T > limit(long maxSize);
```

SplitIterator

```
/**
* An object for traversing and partitioning elements
* of a source.
*
*/
public interface Spliterator< T >;

/**
* Il metodo di avanzamento diventa tryAdvance(), che ribalta il
* funzionamento dell'iterator: non è l'utilizzatore che ottiene il
* nuovo elemento, ma è lo stream che fornisce l'elemento al codice che
* deveoperarci sopra.
*
* If a remaining element exists, performs the given
* action on it, returning true; else returns false.
*
*/
boolean tryAdvance(Consumer< ? super T > action);

/**
* Returns an estimate of the number of elements that
* would be encountered by a forEachRemaining()
* traversal, or returns Long.MAX_VALUE if infinite,
* unknown, or too expensive to compute.
*
* @return the estimated size
*/
long estimateSize();

/**
* notare il tipo di ritorno. Ci si attende che ogni chiamata a questo
* metodo dia lo stesso risultato; eventualmente, il risultato può cambiare dopo
la separazione.
* Returns a set of characteristics of this
* Spliterator and its elements.
*
* @return a representation of characteristics
```

```
*/
int characteristics();

/**
* Performs the given action for each remaining element,
* sequentially in the current thread, until all elements
* have been processed or the action throws an exception.
*
*/
default void forEachRemaining(Consumer< ? super T > action);

/**
* If this spliterator can be partitioned, returns a
* Spliterator covering elements, that will, upon return
* from this method, not be covered by this Spliterator.
*
* @return a Spliterator covering some portion of the
* elements, or null if this spliterator cannot
* be split
*
*/
Spliterator< T > trySplit();
```

Collector

La classe Collectors permette di produrre dei Collector a partire dagli elementi di base:

- un Supplier< A > del contenitore di risultato
- un BiConsumer< A,T > che accumula un elemento nel contenitore
- un BinaryOperator< A > che combina due contenitori parziali
- un Function< A,R > che dal contenitore ottiene il risultato finale

```
/**
* Performs a reduction on the elements of this stream,
* using the provided identity value and an associative
* accumulation function, and returns the reduced value.
*
* @param identity the identity value for the accumulating
* function
* @param accumulator an associative, non-interfering,
* stateless function for combining
* two values
* @result the result of the reduction
*
*/
T reduce(T identity, BinaryOperator< T > accumulator);

/**
* Performs a mutable reduction operation on the elements
* of this stream using a Collector.
*
* @R the type of the result
* @A the intermediate accumulation type of the Collector
* @param the Collector describing the reduction
* @result the result of the reduction
*/
< R,A > R collect(Collector< ? super T,A,R > collector);
```

```
/**
* A mutable reduction operation that accumulates input
* elements into a mutable result container.
*
* @T the type of input elements to the reduction
* operation
* @A the mutable accumulation type of the
* reduction operation
* @R the result type of the reduction operation
*/
interface Collector< T,A,R >;
```