Module java.base
Package java.util.stream

Interface Gatherer<T,A,R>

Type Parameters:

`T` - the type of input elements to the gatherer operation

`A` - the potentially mutable state type of the gatherer operation (often hidden as an implementation detail)

`R` - the type of output elements from the gatherer operation

---

`public interface` **`Gatherer<T,A,R>`**

> **`Gatherer` is a preview API of the Java platform.**
> *Programs can only use `Gatherer` when preview features are enabled.*
> *Preview features may be removed in a future release, or upgraded to permanent features of the Java platform.*

An intermediate operation that transforms a stream of input elements into a stream of output elements, optionally applying a final action when the end of the upstream is reached. The transformation may be stateless or stateful, and may buffer input before producing any output.

Gatherer operations can be performed either sequentially, or be parallelized -- if a combiner function is supplied.

There are many examples of gathering operations, including but not limited to: grouping elements into batches (windowing functions); de-duplicating consecutively similar elements; incremental accumulation functions (prefix scan); incremental reordering functions, etc. The class `Gatherers`$^{\text{PREVIEW}}$ provides implementations of common gathering operations.

API Note:

A `Gatherer` is specified by four functions that work together to process input elements, optionally using intermediate state, and optionally perform a final action at the end of input. They are:

- creating a new, potentially mutable, state (`initializer()`)
- integrating a new input element (`integrator()`)
- combining two states into one (`combiner()`)
- performing an optional final action (`finisher()`)

Each invocation of `initializer()`, `integrator()`, `combiner()`, and `finisher()` must return a semantically identical result.

Implementations of Gatherer must not capture, retain, or expose to other threads, the references to the state instance, or the downstream `Gatherer.Downstream`$^{\text{PREVIEW}}$ for longer than the invocation duration of the method which they are passed to.

Performing a gathering operation with a `Gatherer` should produce a result equivalent to:

```
Gatherer.Downstream<? super R> downstream = ...;
A state = gatherer.initializer().get();
for (T t : data) {
    gatherer.integrator().integrate(state, t, downstream);
}
gatherer.finisher().accept(state, downstream);
```

However, the library is free to partition the input, perform the integrations on the partitions, and then use the combiner function to combine the partial results to achieve a gathering operation. (Depending on the specific gathering operation, this may perform better or worse, depending on the relative cost of the integrator and combiner functions.)

In addition to the predefined implementations in `Gatherers`$^{\text{PREVIEW}}$, the static factory methods `of(...)` and `ofSequential(...)` can be used to construct gatherers. For example, you could create a gatherer that implements the equivalent of `Stream.map(java.util.function.Function)` with:

```
public static <T, R> Gatherer<T, ?, R> map(Function<? super T, ? extends R> mapper) {
    return Gatherer.of(
        (unused, element, downstream) -> // integrator
            downstream.push(mapper.apply(element))
    );
}
```

Gatherers are designed to be *composed*; two or more Gatherers can be composed into a single Gatherer using the `andThen(Gatherer)` method.

```
// using the implementation of `map` as seen above
Gatherer<Integer, ?, Integer> increment = map(i -> i + 1);

Gatherer<Object, ?, String> toString = map(i -> i.toString());

Gatherer<Integer, ?, String> incrementThenToString = increment.andThen(toString);
```

As an example, a Gatherer implementing a sequential Prefix Scan could be done the following way:

```
public static <T, R> Gatherer<T, ?, R> scan(
    Supplier<R> initial,
    BiFunction<? super R, ? super T, ? extends R> scanner) {

    class State {
        R current = initial.get();
    }

    return Gatherer.<T, State, R>ofSequential(
         State::new,
         Gatherer.Integrator.ofGreedy((state, element, downstream) -> {
             state.current = scanner.apply(state.current, element);
             return downstream.push(state.current);
         })
    );
}
```

Example of usage:

```
// will contain: ["1", "12", "123", "1234", "12345", "123456", "1234567", "12345678", "123456789"]
List<String> numberStrings =
    Stream.of(1,2,3,4,5,6,7,8,9)
          .gather(
              scan(() -> "", (string, number) -> string + number)
           )
          .toList();
```

Implementation Requirements:

Libraries that implement transformations based on `Gatherer`, such as `Stream.gather(Gatherer)`$^{\text{PREVIEW}}$, must adhere to the following constraints:

- Gatherers whose initializer is `defaultInitializer()` are considered to be stateless, and invoking their initializer is optional.
- Gatherers whose integrator is an instance of `Gatherer.Integrator.Greedy`$^{\text{PREVIEW}}$ can be assumed not to short-circuit, and the return value of invoking `Gatherer.Integrator.integrate(Object, Object, Downstream)`$^{\text{PREVIEW}}$ does not need to be inspected.
- The first argument passed to the integration function, both arguments passed to the combiner function, and the argument passed to the finisher function must be the result of a previous invocation of the initializer or combiner functions.
- The implementation should not do anything with the result of any of the initializer or combiner functions other than to pass them again to the integrator, combiner, or finisher functions.
- Once a state object is passed to the combiner or finisher function, it is never passed to the integrator function again.
- When the integrator function returns `false`, it shall be interpreted just as if there were no more elements to pass it.
- For parallel evaluation, the gathering implementation must manage that the input is properly partitioned, that partitions are processed in isolation, and combining happens only after integration is complete for both partitions.
- Gatherers whose combiner is `defaultCombiner()` may only be evaluated sequentially. All other combiners allow the operation to be parallelized by initializing each partition in separation, invoking the integrator until it returns `false`, and then joining each partitions state using the combiner, and then invoking the finisher on the joined state. Outputs and state later in the input sequence will be discarded if processing an earlier partition short-circuits.
- Gatherers whose finisher is `defaultFinisher()` are considered to not have an end-of-stream hook and invoking their finisher is optional.

Since:

22

See Also:

`Stream.gather(Gatherer)`$^{\text{PREVIEW}}$, `Gatherers`$^{\text{PREVIEW}}$

Nested Class Summary

Nested Classes

Modifier and Type	Interface	Description
`static interface`	`Gatherer.Downstream`$^{\text{PREVIEW}}$`<T>`	**Preview.** A Downstream object is the next stage in a pipeline of operations, to which elements can be sent.
`static interface`	`Gatherer.Integrator`$^{\text{PREVIEW}}$`<A,T,R>`	**Preview.** An Integrator receives elements and processes them, optionally using the supplied state, and optionally sends incremental results downstream.

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Abstract Methods** | **Default Methods**

Modifier and Type	Method	Description
`default <RR> Gatherer`$^{\text{PREVIEW}}$`<T,?,RR>`	`andThen(Gatherer`$^{\text{PREVIEW}}$`<? super R,?,? extends RR> that)`	Returns a composed Gatherer which connects the output of this Gatherer to the input of that Gatherer.
`default BinaryOperator<A>`	`combiner()`	A function which accepts two intermediate states and combines them into one.
`static <A> BinaryOperator<A>`	`defaultCombiner()`	Returns a combiner which is the default combiner of a Gatherer.
`static <A, R> BiConsumer<A, Gatherer.Downstream`$^{\text{PREVIEW}}$`<? super R>>`	`defaultFinisher()`	Returns a `finisher` which is the default finisher of a `Gatherer`.
`static <A> Supplier<A>`	`defaultInitializer()`	Returns an initializer which is the default initializer of a Gatherer.
`default BiConsumer<A,Gatherer.Downstream`$^{\text{PREVIEW}}$`<? super R>>`	`finisher()`	A function which accepts the final intermediate state and a `Gatherer.Downstream`$^{\text{PREVIEW}}$ object, allowing to perform a final action at the end of input elements.
`default Supplier<A>`	`initializer()`	A function that produces an instance of the intermediate state used for this gathering operation.
`Gatherer.Integrator`$^{\text{PREVIEW}}$`<A,T,R>`	`integrator()`	A function which integrates provided elements, potentially using the provided intermediate state, optionally producing output to the provided `Gatherer.Downstream`$^{\text{PREVIEW}}$.
`static <T, A, R> Gatherer`$^{\text{PREVIEW}}$`<T,A,R>`	`of(Supplier<A> initializer, Gatherer.Integrator`$^{\text{PREVIEW}}$`<A, T,R> integrator, BinaryOperator<A> combiner, BiConsumer<A,Gatherer.Downstream`$^{\text{PREVIEW}}$`<? super R>> finisher)`	Returns a new, parallelizable, `Gatherer` described by the given `initializer`, `integrator`, `combiner` and `finisher`.
`static <T, R> Gatherer`$^{\text{PREVIEW}}$`<T,Void,R>`	`of(Gatherer.Integrator`$^{\text{PREVIEW}}$`<Void,T,R> integrator)`	Returns a new, parallelizable, and stateless `Gatherer` described by the given `integrator`.
`static <T, R> Gatherer`$^{\text{PREVIEW}}$`<T,Void,R>`	`of(Gatherer.Integrator`$^{\text{PREVIEW}}$`<Void,T,R> integrator, BiConsumer<Void,Gatherer.Downstream`$^{\text{PREVIEW}}$`<? super R>> finisher)`	Returns a new, parallelizable, and stateless `Gatherer` described by the given `integrator` and `finisher`.
`static <T, A, R> Gatherer`$^{\text{PREVIEW}}$`<T,A,R>`	`ofSequential(Supplier<A> initializer, Gatherer.Integrator`$^{\text{PREVIEW}}$`<A,T,R> integrator)`	Returns a new, sequential, `Gatherer` described by the given `initializer` and `integrator`.
`static <T, A, R> Gatherer`$^{\text{PREVIEW}}$`<T,A,R>`	`ofSequential(Supplier<A> initializer, Gatherer.Integrator`$^{\text{PREVIEW}}$`<A,T,R> integrator, BiConsumer<A,Gatherer.Downstream`$^{\text{PREVIEW}}$`<? super R>> finisher)`	Returns a new, sequential, `Gatherer` described by the given `initializer`, `integrator`, and `finisher`.
`static <T, R> Gatherer`$^{\text{PREVIEW}}$`<T,Void,R>`	`ofSequential(Gatherer.Integrator`$^{\text{PREVIEW}}$`<Void,T, R> integrator)`	Returns a new, sequential, and stateless `Gatherer` described by the given `integrator`.
`static <T, R> Gatherer`$^{\text{PREVIEW}}$`<T,Void,R>`	`ofSequential(Gatherer.Integrator`$^{\text{PREVIEW}}$`<Void,T, R> integrator, BiConsumer<Void, Gatherer.Downstream`$^{\text{PREVIEW}}$`<? super R>> finisher)`	Returns a new, sequential, and stateless `Gatherer` described by the given `integrator` and `finisher`.

Method Details

initializer

`default Supplier<A> initializer()`

A function that produces an instance of the intermediate state used for this gathering operation.

Implementation Requirements:

The implementation in this interface returns `defaultInitializer()`.

Returns:

A function that produces an instance of the intermediate state used for this gathering operation

integrator

`Gatherer.Integrator`$^{\text{PREVIEW}}$`<A,T,R> integrator()`

A function which integrates provided elements, potentially using the provided intermediate state, optionally producing output to the provided `Gatherer.Downstream`$^{\text{PREVIEW}}$.

Returns:

a function which integrates provided elements, potentially using the provided state, optionally producing output to the provided Downstream

combiner

`default BinaryOperator<A> combiner()`

A function which accepts two intermediate states and combines them into one.

Implementation Requirements:

The implementation in this interface returns `defaultCombiner()`.

Returns:

a function which accepts two intermediate states and combines them into one

finisher

`default BiConsumer<A,Gatherer.Downstream`$^{\text{PREVIEW}}$`<? super R>> finisher()`

A function which accepts the final intermediate state and a `Gatherer.Downstream`$^{\text{PREVIEW}}$ object, allowing to perform a final action at the end of input elements.

Implementation Requirements:

The implementation in this interface returns `defaultFinisher()`.

Returns:

a function which transforms the intermediate result to the final result(s) which are then passed on to the provided Downstream

andThen

`default <RR> Gatherer`$^{\text{PREVIEW}}$`<T,?,RR> andThen(Gatherer`$^{\text{PREVIEW}}$`<? super R,?,? extends RR> that)`

Returns a composed Gatherer which connects the output of this Gatherer to the input of that Gatherer.

Implementation Requirements:

The implementation in this interface returns a new Gatherer which is semantically equivalent to the combination of `this` and `that` gatherer.

Type Parameters:

`RR` - The type of output of that Gatherer

Parameters:

`that` - the other gatherer

Returns:

returns a composed Gatherer which connects the output of this Gatherer as input that Gatherer

Throws:

`NullPointerException` - if the argument is `null`

defaultInitializer

`static <A> Supplier<A> defaultInitializer()`

Returns an initializer which is the default initializer of a Gatherer. The returned initializer identifies that the owner Gatherer is stateless.

Implementation Requirements:

This method always returns the same instance.

Type Parameters:

`A` - the type of the state of the returned initializer

Returns:

the instance of the default initializer

See Also:

`initializer()`

defaultCombiner

`static <A> BinaryOperator<A> defaultCombiner()`

Returns a combiner which is the default combiner of a Gatherer. The returned combiner identifies that the owning Gatherer must only be evaluated sequentially.

Implementation Requirements:

This method always returns the same instance.

Type Parameters:

`A` - the type of the state of the returned combiner

Returns:

the instance of the default combiner

See Also:

`finisher()`

defaultFinisher

`static <A, R>`
`BiConsumer<A,Gatherer.Downstream`$^{\text{PREVIEW}}$`<? super R>> defaultFinisher()`

Returns a `finisher` which is the default finisher of a `Gatherer`. The returned finisher identifies that the owning Gatherer performs no additional actions at the end of input.

Implementation Requirements:

This method always returns the same instance.

Type Parameters:

`A` - the type of the state of the returned finisher

`R` - the type of the Downstream of the returned finisher

Returns:

the instance of the default finisher

See Also:

`finisher()`

ofSequential

`static <T, R> Gatherer`$^{\text{PREVIEW}}$`<T,Void,R> ofSequential(Gatherer.Integrator`$^{\text{PREVIEW}}$`<Void,T,R> integrator)`

Returns a new, sequential, and stateless `Gatherer` described by the given `integrator`.

Type Parameters:

`T` - the type of input elements for the new gatherer

`R` - the type of results for the new gatherer

Parameters:

`integrator` - the integrator function for the new gatherer

Returns:

the new `Gatherer`

Throws:

`NullPointerException` - if the argument is `null`

ofSequential

`static <T, R> Gatherer`$^{\text{PREVIEW}}$`<T,Void,R> ofSequential(Gatherer.Integrator`$^{\text{PREVIEW}}$`<Void,T,R> integrator,`
`BiConsumer<Void,Gatherer.Downstream`$^{\text{PREVIEW}}$`<? super R>> finisher)`

Returns a new, sequential, and stateless `Gatherer` described by the given `integrator` and `finisher`.

Type Parameters:

`T` - the type of input elements for the new gatherer

`R` - the type of results for the new gatherer

Parameters:

`integrator` - the integrator function for the new gatherer

`finisher` - the finisher function for the new gatherer

Returns:

the new `Gatherer`

Throws:

`NullPointerException` - if any argument is `null`

ofSequential

`static <T, A, R> Gatherer`$^{\text{PREVIEW}}$`<T,A,R> ofSequential(Supplier<A> initializer,`
`Gatherer.Integrator`$^{\text{PREVIEW}}$`<A,T,R> integrator)`

Returns a new, sequential, `Gatherer` described by the given `initializer` and `integrator`.

Type Parameters:

`T` - the type of input elements for the new gatherer

`A` - the type of state for the new gatherer

`R` - the type of results for the new gatherer

Parameters:

`initializer` - the initializer function for the new gatherer

`integrator` - the integrator function for the new gatherer

Returns:

the new `Gatherer`

Throws:

`NullPointerException` - if any argument is `null`

ofSequential

`static <T, A, R> Gatherer`$^{\text{PREVIEW}}$`<T,A,R> ofSequential(Supplier<A> initializer,`
`Gatherer.Integrator`$^{\text{PREVIEW}}$`<A,T,R> integrator,`
`BiConsumer<A,Gatherer.Downstream`$^{\text{PREVIEW}}$`<? super R>> finisher)`

Returns a new, sequential, `Gatherer` described by the given `initializer`, `integrator`, and `finisher`.

Type Parameters:

`T` - the type of input elements for the new gatherer

`A` - the type of state for the new gatherer

`R` - the type of results for the new gatherer

Parameters:

`initializer` - the initializer function for the new gatherer

`integrator` - the integrator function for the new gatherer

`finisher` - the finisher function for the new gatherer

Returns:

the new `Gatherer`

Throws:

`NullPointerException` - if any argument is `null`

of

`static <T, R> Gatherer`$^{\text{PREVIEW}}$`<T,Void,R> of(Gatherer.Integrator`$^{\text{PREVIEW}}$`<Void,T,R> integrator)`

Returns a new, parallelizable, and stateless `Gatherer` described by the given `integrator`.

Type Parameters:

`T` - the type of input elements for the new gatherer

`R` - the type of results for the new gatherer

Parameters:

`integrator` - the integrator function for the new gatherer

Returns:

the new `Gatherer`

Throws:

`NullPointerException` - if any argument is `null`

of

`static <T, R> Gatherer`$^{\text{PREVIEW}}$`<T,Void,R> of(Gatherer.Integrator`$^{\text{PREVIEW}}$`<Void,T,R> integrator,`
`BiConsumer<Void,Gatherer.Downstream`$^{\text{PREVIEW}}$`<? super R>> finisher)`

Returns a new, parallelizable, and stateless `Gatherer` described by the given `integrator` and `finisher`.

Type Parameters:

`T` - the type of input elements for the new gatherer

`R` - the type of results for the new gatherer

Parameters:

`integrator` - the integrator function for the new gatherer

`finisher` - the finisher function for the new gatherer

Returns:

the new `Gatherer`

Throws:

`NullPointerException` - if any argument is `null`

of

`static <T, A, R> Gatherer`$^{\text{PREVIEW}}$`<T,A,R> of(Supplier<A> initializer,`
`Gatherer.Integrator`$^{\text{PREVIEW}}$`<A,T,R> integrator,`
`BinaryOperator<A> combiner,`
`BiConsumer<A,Gatherer.Downstream`$^{\text{PREVIEW}}$`<? super R>> finisher)`

Returns a new, parallelizable, `Gatherer` described by the given `initializer`, `integrator`, `combiner` and `finisher`.

Type Parameters:

`T` - the type of input elements for the new gatherer

`A` - the type of state for the new gatherer

`R` - the type of results for the new gatherer

Parameters:

`initializer` - the initializer function for the new gatherer

`integrator` - the integrator function for the new gatherer

`combiner` - the combiner function for the new gatherer

`finisher` - the finisher function for the new gatherer

Returns:

the new `Gatherer`

Throws:

`NullPointerException` - if any argument is `null`

Report a bug or suggest an enhancement
For further API reference and developer documentation see the Java SE Documentation, which contains more detailed, developer-targeted descriptions with conceptual overviews, definitions of terms, workarounds, and working code examples. Other versions.