**Stream API** – Набор классов и интерфейсов, представляющий из себя поток однотипных данных и выполнение над ними однотипных операций.

Представлен 2 видами интерфейсов-наследников BaseStream:

*Stream* – интерфейс для работы с объектными данными.  
*IntStream, DoubleStream, LongStream* – интерфейсы для эффективной работы с примитивными типами данных (во избежание лишней упаковки в объекты).

\*Реализации этих интерфейсов лежат в соответствующих Pipline классах.

**Виды методов:**

1. Операции над источником
2. Промежуточные
3. Терминальные

|  |  |  |
| --- | --- | --- |
| Источник | | |
| Stream.empty() | Создает пустой стрим | Возвращает не какой-то инстанс (один и тот же объект), а всегда новый пустой стрим, поскольку у стримов есть состояние о том был ли он использован или нет. |
| Stream.of () | Стрим из набра объектов |  |
| Arrays.stream() | Стрим из массива |  |
| Stream.ofNullable() | Принимает один элемет | Возвращает пустой стрим, если передан null |
| Stream.generate() | Создает бесконечный стрим | Принимает Supplier |
| Intstream.iterate() | Создает бесконечный стрим, где каждый следующий элемент вычисляется из предыдущего |  |
| collection.stream() | Создает стрим из коллекции |  |
| IntStream.range | Стрим диапазона |  |
| new Random /inits /longs /doubles | Стрим из соответствующих чисел | Без параметров – бесконечный стрим. С помощью параметров можно задать размер и диапазон чисел |
| String.chars() | Поток int | Поток символов не описан в API, поэтому возвращается IntStream |

|  |  |  |
| --- | --- | --- |
| Промежуточные | | |
| map – возвращает Object  /mapToInt /mapToLong /mapToDouble  – возвращают соответствующий  Stream (не будет лишнего боксинга) | Изменяет тип входного значения |  |
| filter | Пропускает элементы удовлетворяющие условию |  |
| flatMap  /flatMapToInt /flatMapToLong /flatMapToDouble | Возвращает некоторое кол-во подстримов из исходного стрима |  |
| distinct | Пропускает уникальные элементы | Stateful |
| sorted | Сортирует | Stateful и full barrier |
| Limit | Ограничивает стрим по аргументу | Операция с коротким замыканием (если передается бесконечный стрим, то он будет конечным) |
| skip | Пропускает элементы стрима по элементу |  |
| Peek | Заглядывает в текущий элемент стрима (для отладки) |  |
| takeWhile | Java 9 – работает, пока выполняется условие, как только первый элемент не удовлетворяет условию – перестает обращаться к источнику | Операция с коротким замыканием |
| dropWhile | Java 9 – Не пропускает элементы до тез пор, пока условие не будет выполнено, после этого не проверяет элементы. |  |
| Boxed /asLongStream /asDoubleStream | Преобразуют примитивные стримы в объектные стримы |  |
| unordered  parallel  sequential | Используются для конфигурирования поведения стрима путем выставления на нем соответствующих флагов |  |

\*Стандартное поведение стрима – каждый элемент стрима проходит всю цепь промежуточных методов до терминального, после этого обрабатывается следующий элемент (если нет sorted()).

*Statful* – методы сохраняющие состояние элементов ранее прошедших через него.  
Stateless – не сохраняют состояния предыдущих элементов.

*Full barrier* – изменяет стандартное поведение стрима. Такой метод аккомулирует в себе все элементы стрима и только после соблюдается стандартное поведение.

|  |  |  |
| --- | --- | --- |
| Терминальные операции | | |
| Count | Считает кол-во элементов | если между ним и стримом нет методов, которые в перспективе могли бы изменить стрим, то эти промежуточные методы не отработают |
| findFirst /findAny | Возвращают первый элемент стрима | Для параллельных стримов оверхед на findFirst, т.к. нужно синхронизировать результаты параллельных потоков и вернуть истино первый результат. |
| AnyMatch /noneMatch /allMatch | возвращает boolean, семантика соответствует названию метода |  |
| forEach /forEachOrdered | выполнить действие для всех элементов | forEach не гарантирует порядок при использовании в параллельных стримах, ordered – гарантирует |
| Max /min | возвращает максимальный, минимальный элемент коллекции |  |
| Sum /average /summaryStatistics | Методы примитивных стримов – sum – складывает все элементы стрима, average – среднее арифметическое, summary – статистика по стриму |  |
| toArray | Собрать стрим в массив |  |
| reduce | Сводит поток элементов в один результат (использовать в случае, если не подходят стандартные терминальные операции). |  |
| collect | предоставляет шаблоны комбинирования объектов в одно целое | Принимает класс Collector или определение кастомных функций. |

**Reduce**:

Optional<T> reduce(BinaryOperator<T> accumulator) – требует только функцию, которая будет аккумулировать каждый элемент.

T reduce(T identity, BinaryOperator<T> accumulator) – identity – начальное значение или значение по умолчанию, если элементов нет.

U reduce(U identity, BiFunction<U,? super T,U> accumulator, BinaryOperator<U> combiner) – 3й параметр – функция, которая будет складывать результаты параллельных стримов, нет смысла, если стрим не параллельный

**Collectors** – Утилитарный класс, представляющих из себя набор статических методов-шаблонов, которые определяют поведение для аккомулирования результатов в нужную структуру.

|  |  |
| --- | --- |
| toList(), toSet() | Собирает Stream в список или сет – не гарантирует реализацию, но стандартно возвращает ArrayList и HashSet |
| toCollection(Supplier) | Если нужно собрать в конкретную реализацию коллекций |
| toMap() | Собирает в Map, принимает 4 параметра Function, для ключа и для значения ключа, 3я функция это поведение при повторяющемся ключе, 4я функция Supplier – можно передать нужную коллекцию |
| Joining() | для склеивания строк, если необходимо принимает разделитель строк, префикс и суффикс. |
| groupingBy() | 1 параметр – классификатор для группировки (ключ), при использовании 1 параметра значения будут сгруппированы в Map, значением каждого ключа будет List<> 2 параметра – классификатор, downstream – рецепт склеивания значений в 1 объект, значением каждого ключа будет обычный объект. |
| Вспомогательные коллекторы | Коллекторы, дублирущие методы стрима (counting, mapping и др.), необходимы, если нужно сделать дополнительные преобразования с элементами перед их сбором в конечную структуру. |

Пример замены цикла на Stream с фиксированным числом «итераций»:

String[] chars = s.split("");  
return IntStream.*range*(0, s.length())  
 .mapToObj(i->chars[i].toUpperCase() + chars[i].toLowerCase().repeat(i) )  
 .collect(Collectors.*joining*("-"));

**Особенности:**

1. Стрим является способом обработки данных, а не их представлением (не хранит данные, за некоторым исключением), в качестве способа обработки используется обход элементов с помощью Spliterator.

Spliterator – интерфейс по функционалу схожий с Iterator:

*Сходства*: предоставляет возможность последовательного обхода всех элементов, одноразовый, возлагает реализацию на источник данных.

*Отличия:* предоставляет возможность разделения источника данных на части, что делает возможных их параллельную обработку, отсутствие возможности изменения источника, отмечает состояния источника для оптимизации работы Stream (ordered, distinct, sorted, sized, nonnull, immutable, concurrent, subSized).

1. Ленивый – непосредственное исполнение Stream происходит при вызове терминальной операции. Операции над источником и промежуточные операции формируют последовательность шагов (возвращают Stream), которая отражается на поведении терминальной операции, то есть формируют последовательность шагов, а терминальная операция работает с этой последовательностью.

**Пралельные стримы:**

Spliterator предварительно разбивает данные на фрагменты, которые передаются в ForkJoinPool, который выделяет и хранит потоки для исполнения этих задач и координирует их работу.

**Optional и null**

Null – ключевое слово для обозначения отсутствия значения у ссылочного типа. Семантика null предполагает обозначение неинициализированного состояния переменной (переменная еще не подгрузилась, либо вовсе не была инициализирована). Проблема заключается в том, что это состояние доступно не только компилятору, но и программисту – с помощью присваивания ссылочным типам null (Null reference – the billion dollar mistake). Разрешение к присвоению ссылочным типам null привело к многочисленным проблемам. Из-за null reference значение null стало более размыто, т.к. программист мог присвоить null значению иную семантику (например, если пользователь не ввел имя, присвоить имени null, как обозначение его отсутствия), наличие NPE и размытой семантики null обязывало постоянно обрабатывать возможные ситуации с NPE.

Борьба с NPE:

1) Вообще не использовать null в коде  
2) Не оставлять поля пустыми, инициализировать его прямо в классе (если это универсальное поле)   
3) Использовать в коде проверки типа if (a==null)  
4) Использование аннотаций типа NotNull (компилятором создается проверка на null – не надо писать руками, в случае аргумента == null вылетает IlligalArgumentException) и Nullable (указывает, что аргумент в метода может принимать null)  
5) Использование ключ-слова final  
6) Использование паттерна Null Object

7)Использование класса Optional, если null является ожидаемым результатом метода.

*Optional* – контейнер из одного элемента, который может хранить значение (не null), либо отсутствие значения.

Проблема в Java: в дизайне языка заложено, что любое возвращаемое значение ссылочного типа может быть возвращено методом в виде этого типа, либо в виде null. То есть любое значение ссылочного типа nullable.

Optional используется в качестве возвращаемого значения, чтобы сигнализировать использующему, что тип возвращаемого значения неоднозначен, что в коде присутствуют ситуации, при которых возвращаемый тип может быть null.

То есть класс используется для расширения типов возвращаемых значений.

**Создание объекта**:

1. Optional.of(obj) – создает обертку над объектом в виде Optional со значением переданного аргумента – нельзя передать null (NPE).  
   of следует использовать, если есть уверенность, что в логике программы не может быть null, появление которого означает, что в программе баг (поведение, которого там не должно быть), то есть это реальная исключительная ситуация, обернув которую в пустой Optional будет невозможно отследить.
2. Optional.ofNullable(obj) – делает тоже самое, но если значение аргумента null, возвращает Optional.empty()  
   Говорит о том, что null является ожидаемым результатом.
3. Optional.empty() – возвращает Optional, значение которого null

Имеет методы схожие со стримами:

filter (Predicate pr) – возвращает Optional, удовлетворяющий условию или Optional.empty()

map (Function fu) – возвращает Optional с измененным значением, при этом, если null, то возвращается Optional.empty().

flatMap – смысл – вложенную структуру сделать плоской, позволяет избавиться от ситуации, когда есть вложенные Optional, удобно использовать c методами, возвращающими Optional, пример:

orElse(T t) – если значение равно Optional.isEmpty, то возвращает заданное значение.

orElseGet(Supplier sp) – делает тоже самое, что и метод orElse, используется если возвращаемое значение нужно вычислить.

orElseThrow () – если Optional.isEmpty() кидает заданное исключение.

Or (Supplier sp) – если Optional.isEmpty(), возвращает другой Optional, определенный в функции Supplier

isPresent/isEmpty () – проверка Optional