1. 流
2. 概述

与集合相比，流提供了一种可以让我们在更高的概念级别上指定计算任务的数据视图，通过使用流，只需要说明想要完成什么任务，而不用说明如何实现它。流遵循的是“做什么而非怎么做”的原则，不需要指定该操作应该以什么顺序或者在哪个线程中执行。

1. 流与集合的区别
2. 流并不存储元素，这些元素可能存储在底层的集合中，或者是按需生成的；
3. 流的操作不会修改其数据源，而是生成一个新的流；
4. 流的操作尽可能是惰性执行的，直至需要其结果时，操作才会执行。
5. 操作流的典型流程

包含三个阶段的操作管道：

1. 创建一个流；
2. 指定将初始流转换为其他流的中间操作，可能包含多个步骤；
3. 应用停止操作，从而产生结果，自此之后，这个流无法继续使用了。
4. 流的创建
5. Stream.of(T… values)

产生一个元素为给定值的流，且参数是可变长的。如下所示：

Stream<String> song = Stream.of(“gently”, “down”);

1. Stream.empty()

产生一个不包含任何元素的流。

Stream<String> silence = Stream.empty();

1. Stream.generate(Supplier<T> s)

产生一个无限流，它的值是通过反复调用函数s，可以说是lambda表达式构建的。

Stream<String> echos = Stream.generate(()->”echo”);

1. Stream.iterate(T seed, UnaryOperator<T> f)

产生一个无限流，接受一个种子值seed，以及一个函数，反复地将该函数应用到之前的结果上。如要产生一个像1,2,3…这样的无限序列，就额可以通过iterate方法：

Stream<BigInteger> integers = Stream.iterate(BigInteger.ZERO, n -> n.add(BigInteger.ONE));

1. Stream.iterate(T seed, Predicate<? super T> hasNext, UnaryOperator<T> f)

比上面的中间多了一个谓词参数，用于判断迭代应该在满足什么条件下结束，只要该谓词拒绝了某个迭代生成的值，这个流就会结束。

var limit = new BigInteger(“100000”);

Stream<BigInteger> integers = Stream.iterate(BigInteger.ZERO, n -> n.compareTo(limit) > 0, n->n.add(BigInteger.ONE));

1. Stream.ofNullable(T t)

用一个对象来创建一个很短的流，如果该对象为null， 则流的长度为0；否则，流的长度为1，即只包含该对象。

Stream.ofNullable(limit);

1. Arrays.stream(T[] array, int startInclusive, int endExclusive)

产生一个流，元素由数组中指定范围内的元素构成。

List<String> words = List.of(“a”, “b”);

long count = words.stream().filter(w -> w.length() > 12).count();

不同的类有不同的产生流的方式，这里就不一一介绍了。

1. 流的转换
2. 概述

流的转换会产生一个新的流，它的元素派生自另一个流中的元素。

1. Stream.filter(Predicate<? super T> predicate)

产生一个流，它包含当前流中所有满足谓词条件的元素。

List<String> words = …;

Stream<String> long = words.stream().filter(w -> w.length()>12);

filter中的引元是Predicate<T>，即从T到Boolean的函数。

1. Stream.map(Function<? super T, ? extends R> mapper)

产生一个流，包含将mapper函数应用于当前流中所有元素所产生的结果。如将所有单词都转换成小写：

Stream<String> lower = words.stream().map(String::toLowerCase);

Stream<String> lower = words.stream().map(s -> s.substring(0, 1));

1. Stream.flatMap(Function<? super T, ? extends Stream<? extends R>> mapper)

产生一个流，通过将mapper函数应用于当前流中的所有元素所产生的结果，并且将这些结果连接在一起而获得的，每个结果都是一个流，最终连接得到一个流。

1. Stream.limit(long n)

产生一个流，其中包含了当前流中最初的n个元素，经常应用于裁剪无限流的尺寸：

Stream<Double> randoms = Stream.generate(Math::random).limit(100);

1. Stream.skip(long n)

产生一个流，包含当前流中除了前n个元素以外的所有元素。

1. Stream.takeWhile(Predicate<? super T> predicate)

产生一个流，它的元素是当前流中所有满足谓词条件的元素，在谓词为真时获取流中的元素，然后停止。

1. Stream.dropWhile(Predicate<? super T> predicate)

产生一个流，它的元素是当谓词为真时丢弃满足谓词条件的元素，并产生一个由第一个使该谓词为假的字符开始的所有元素。

1. Stream.concat(Stream<? extends T> a, Stream<? extends T> b)

产生一个流，将流a和流b连接起来，注意，a一定不能是无限的，不然流b永远都无法连接上去。

1. Stream.distinct()

产生一个流，将原来的流的元素按照相同顺序剔除重复元素。

1. Stream.sorted(Comparator<? super T> comparator)

产生一个流，按照排序器的排序规则排序，如对字符串排序，使得最长的字符串排在最前面：

Stream<String> long = words.stream().sorted(Comparator.comparing(String::length).reversed());

1. Stream.peek(Consumer<? super T> action)

产生一个流，与当前流中的元素相同，但是在每次获取一个元素时，都会调用函数action。

Object[] powers = Stream.iterate(1.0, p -> p\*2).peek(e ->System.out.println(e)).limit(29).toArray();

1. 流的约简
2. 概述

从流数据中获取答案的方式称为约简，是一种终结操作，将流约简为可以在程序中使用的非流值。

1. Stream.max(Comparator<? super T> comparator)

产生这个流的最大元素，使用比较器的排序原则，如果这个流为空，则产生一个空的Optional对象。

1. Stream.min(Comparator<? super T> comparator)

产生这个流的最小元素，使用比较器的排序原则，如果这个流为空，则产生一个空的Optional对象。

1. Stream.findFirst()

返回流中的第一个值。

Optional<String> starts = words.filter(s->s.startsWith(“Q”)).findFirst();

1. Stream.findAny()

返回这个流中的任意元素，通常用于并行处理流，因为可以任意地匹配。

1. Stream.anyMatch(Predicate<? super T> predicate)

流中任意元素能够匹配给定的谓词时返回true。

1. Stream.allMatch(Predicate<? super T> predicate)

流中所有元素都匹配谓词时返回true。

1. Stream.noneMatch(Predicate<? super T> preidicate)

流中没有元素匹配谓词时返回true。