# 基本概念

Java Stream 是 Java 8 引入的用于处理集合数据的新特性。它提供了一种更简洁、更高效的方式来操作和处理集合中的元素。

# Stream接口

## 方法

### Optional<T> findAny();

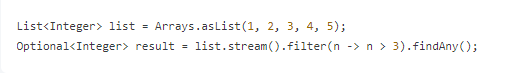
用于在流中查找某个元素并返回一个 Optional 对象。如果当前流为空，则返回一个空的 Optional 对象。

有两种情况下可以使用 findAny() 方法：

1、当我们不关心返回的元素是哪一个时，可以使用该方法进行查找，因为 findAny() 方法会随机地返回流中的任意一个元素。

2、当需要查找流中的某个元素，但是又不想遍历整个流时，可以使用该方法，因为它只需要返回任意一个符合条件的元素即可。

示例：



上面的代码中，我们首先创建了一个包含 1 到 5 的整数列表，然后调用 stream() 方法将列表转换成一个 Stream 流。接着，使用 filter() 方法过滤出大于 3 的元素，并使用 findAny() 方法查找该流中的任意一个元素。最终结果为一个 Optional 对象，其中包含值为 4 或 5 中的一个。需要注意的是，findAny() 方法在使用并行流时会更加有效。因为在并行操作中，多个线程可以同时查找流中的元素，从而提高性能。

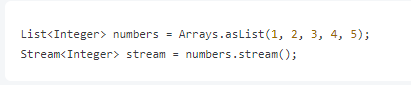
# Collectors类

## 方法

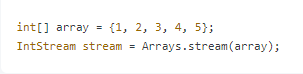
### public static <T> Collector<T, ?, List<T>> toList()

# 创建Stream

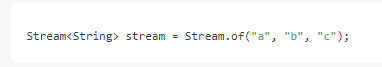
## 1、通过集合创建



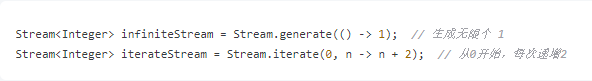
## 2、通过数组创建



## 3、通过Stream.of()方法创建

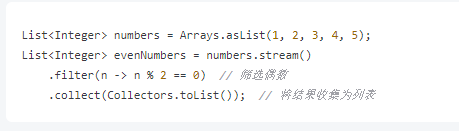


## 4、通过 Stream.generate() 或 Stream.iterate() 方法创建无限流



# 过滤操作

## filter()方法

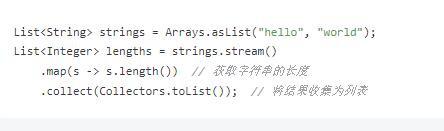


# 映射操作

## map()方法

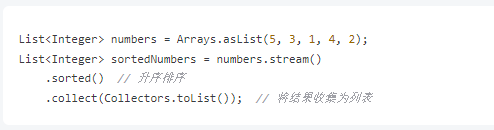
map() 方法是 Java 8 中 Stream 接口提供的一个中间操作方法，用于对流中的每个元素执行指定的映射操作，并将映射后的结果组成一个新的流返回。它能够以简洁的方式对流中的元素进行映射和转换，生成新的流。通过 map() 方法，我们可以方便地进行诸如类型转换、属性提取、数据计算等操作，使得代码更加清晰、简洁。

## 代码示例



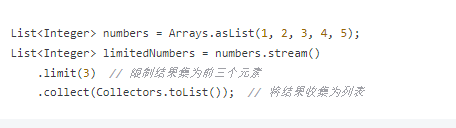
# 排序操作

## sorted()



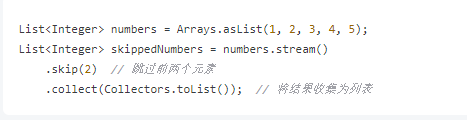
# 限制结果集操作

## limit()



# 跳过元素

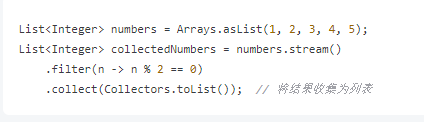
## skip()



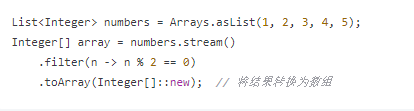
聚合操作

# 终端操作

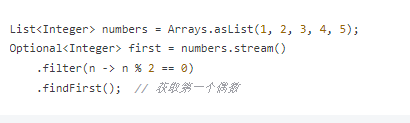
## 收集结果



## 转换为数组



## 获取首个元素



## 检查匹配

