

# Stream API

Stream API是Java 8中引入的一种新的API，它提供了一种用流的方式对集合（Collection）和数组（Array）进行处理的方法，它的目的是使得代码更加简洁、易读和易于维护。

Stream API的主要优点包括：

1. 方便：使用Stream API可以方便地处理集合或数组中的元素，无需手动写循环语句。
2. 简洁：使用Stream API可以用更少的代码完成相同的任务，因为Stream API提供了一系列的函数式操作，例如过滤、映射、排序等，这些操作可以链式调用，从而使代码更加简洁易懂。
3. 高效：使用Stream API可以提高代码的执行效率，因为Stream API使用了惰性求值的方式，只有在需要时才会执行操作，避免了不必要的计算和内存占用。

Stream API的核心概念包括：

1. 流（Stream）：Stream是一种序列化的对象集合，它支持顺序和并行两种操作模式，并且可以在不修改原始集合的情况下进行各种操作。
2. 操作（Operations）：Stream提供了一系列的函数式操作，例如过滤、映射、排序等，这些操作可以链式调用，形成操作链。
3. 终止操作（Terminal Operations）：最后必须使用终止操作，例如forEach、count、reduce等，才能得到操作的结果。

Stream API的出现使得Java在集合处理方面更加强大和灵活，也使得Java代码变得更加简洁、易读和易于维护。

使用Stream API，可以通过一系列的中间操作对集合或数组进行处理，最后再通过终止操作获取结果。下面是一些使用Stream API的示例：

## 1. 集合过滤：

假设有一个List<String>类型的集合，需要在需要过滤出其中长度大于3的元素，可以使用如下代码：

```
List<String> list = Arrays.asList("apple", "banana", "cat", "dog", "elephant");
List<String> filteredList = list.stream()
    .filter(str -> str.length() > 3)
    .collect(Collectors.toList());
System.out.println(filteredList); // [apple, banana, elephant]
```

在上面的代码中，我首先用stream()方法将集合转换成Stream对象，然后使用filter操作过滤出长度大于3的元素，最后使用collect操作将结果转换成List类型。

## 2. 集合映射：

假设有一个List<Integer>类型的集合，需要在需要其中每个元素乘以2，可以使用如下代码：

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);
List<Integer> mappedList = list.stream()
    .map(num -> num * 2)
    .collect(Collectors.toList());
System.out.println(mappedList); // [2, 4, 6, 8, 10]
```

在上面的代码中，我首先用stream()方法将集合转换成Stream对象，然后使用map操作对每个元素进行乘以2的操作，最后使用collect操作将结果转换成List类型。

## 3. 数组排序：

假设有一个int[]类型的数组，需要在需要其中的元素按照从小到大的顺序进行排序，可以使用如下代码：

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);
List<Integer> mappedList = list.stream()
    .map(num -> num * 2)
    .collect(Collectors.toList());
System.out.println(mappedList); // [2, 4, 6, 8, 10]
```

在上面的代码中，我首先用Arrays.stream()方法将数组转换成IntStream对象，然后使用sorted操作对元素进行排序，最后使用toArray操作将结果转换成int[]类型。

