```
    .forEach(e-> {
        System,out printin(e)
    });
```

执行上述代码，将输出：

```
0
2
4
6
8
10
12
14
16
18
```

> 注意：由于`Stream.iterate()`生成的是无限序列流。因此我们应该定义终止条件，例如：limit、findFirst 或 findAny 等，以避免无限循环。

3. 集合转换：`collectingAndThen()`

`collectingAndThen()`方法是在 Java 8 中引入的。它是一种特殊的收集器，允许您对另一个收集器的结果执行特殊类型的转换。

在下面的示例中，我们的收集器通过首先使用索引到大写操作进行映射，然后使该映射成为不可修改的`Map`进行转换。

```
List<String> fruits = Arrays.asList(”apple”, ”banana”,”orange”);
```

```
Map<Integer, String> result = fruits.stream()
  .collect(Collectors.collectingAndThen(
    Collectors.toMap(fruits::index0f,String::toUpperCase),
    Collections::unmodifiableMap
  ));
System.out.println(result)
```

执行上述代码，将输出：

```
{0=APPLE, 1=BANANA, 2=ORANGE}
```

```
Stream.concat(stream1, stream2)
  .forEach(System.out::println);
```

执行上述代码，将输出：

```
1
2
3
4
5
6
```

8. 分组：`Collectors.partitioningBy`

`Collectors.partitioningBy`可以用来对流进行分组。

在下面的示例中，我们根据元素的字符串长度分为两个不同的组。

```
```

```
Map<Boolean, List<String>> result1 = Stream.of(...values: ”apple”, ”banana”, ”orange”, ”grape”)
  .collect(Collectors.partitioningBy(f -> f.length() > 5));

System.out.println(result1);
```

执行上述代码，将输出：

```
{false=[apple, grape], true=[banana, orange]}
```

今天的分享就到这里。如果您学习过程中如遇困难？可以加入我们超高质量的[技术交流群](http://cxyroad.com/”https://www.didispace.com/jiaqun.html”)，参与交流与讨论，更好的学习与进步

> 欢迎我的公众号：程序猿DD。第一时间了解前沿行业消息、分享深度技术干货、获取优质学习资源

原文链接: https://juejin.cn/post/7376213569443545099