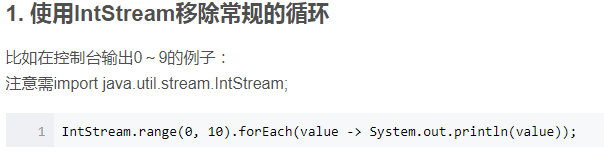
**1.Java 8 Stream**

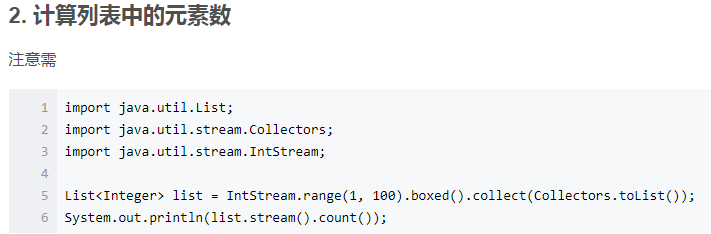
Stream API引入的目的在于弥补Java函数式编程的缺陷。对于很多支持函数式编程的语言，map()、reduce()基本上都内置到语言的标准库中了，不过，Java 8的Stream API总体来讲仍然是非常完善和强大，足以用很少的代码完成许多复杂的功能。

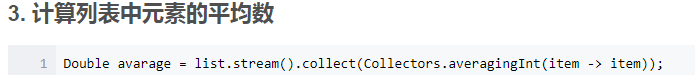
Java 8的Stream API充分利用Lambda表达式的特性，极大的提高编程效率和程序可读性。同时它提供串行和并行两种模式进行汇聚操作，并发模式能够充分利用多核处理器的优势，使用fork/join并行方式来拆分任务和加速处理过程。通常编写并行代码很难而且容易出错，但使用Stream API无需编写一行多线程的代码，就可以很方便地写出高性能的并发程序。

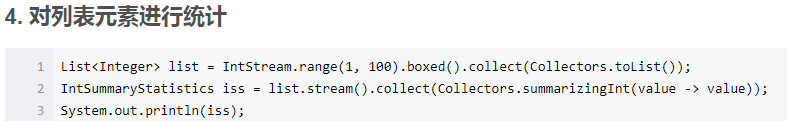
在Stream API中，一个流基本上代表一个元素序列，Stream API提供了丰富的操作函数来计算这些元素。以前我们在开发业务应用时，通常很多操作的实现是这样做的：我们使用循环对集合做遍历，针对集合中的元素实现各种操作，定义各种变量来实现目的，这样我们就得到了一大堆丑陋的顺序代码。

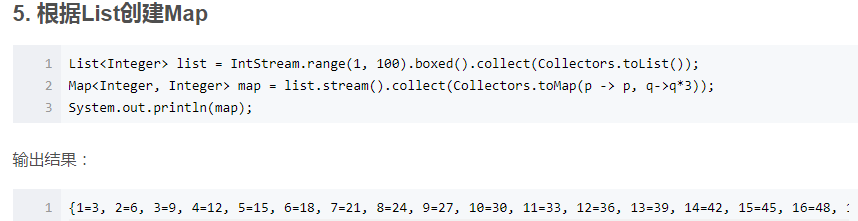
如果我们使用Stream API做同样的事情，使用Lambda表达式和其它函数进行抽象，可以使得代码更易于理解、更为干净。有了这些抽象，还可以做一些优化，比如实现并行等。

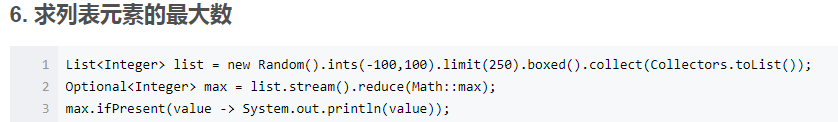












clipboard.png

String[] names = { "Fred Edwards", "Anna Cox", "Deborah Patterson", "Ruth Torres", "Shawn Powell",
"Rose Thompson", "Rachel Barnes", "Eugene Ramirez", "Earl Flores", "Janice Reed", "Sarah Miller",
"Patricia Kelly", "Carl Hall", "Craig Wright", "Martha Phillips", "Thomas Howard", "Steve Martinez",
"Diana Bailey", "Kathleen Hughes", "Russell Anderson", "Theresa Perry" };
List<String> ls = Arrays.asList(names).stream().filter(s -> s.startsWith("C")).collect(Collectors.toList());
System.out.println(ls.toString());

