JDK1.8的优点：

速度快

jdk1.8哈希表增加了红黑数数据结构

代码更少

Lambda表达式\*1

Stream API

便于并行

最大化减少空指针异常：Optional

Nashorn引擎，允许在JVM上运行JS应用

\*1

Lambda表达式：

创建接口的实现类的方法：

通过new接口实现类创建对象；

匿名内部类创建接口实现类对象：Test test = new Test(){[代码块里实现接口的方法]};

Lambda表达式实现接口方法：Test test = () -> [接口的实现方法];

-->箭头操作符，又叫Lambda操作符

左侧对应形参列表，只有单个参数可以省略小口号

右侧对应重写方法方法体的具体内容，只有一个方法时可以省略大括号；

Lambda表达式使用要求，必须是只有一个抽象方法的接口。

只有一个抽象方法的接口，叫做函数式接口。编译期注解@FunctionalInterface检测该接口是否为函数式接口，不是会报错

JDK8以后可以在接口中定义非抽象方法

jdk1.8新增内置函数式接口包java.util.function：

Consume消费型接口void accept(T t);传入参数没有返回值类型

Function函数形R apply(T t)；传入T返回一个R

Predicate断定型接口boolean test(T t)传入T返回值为布尔类型

Supplier供给型接口T get()不用传入参数，有返回值

BiPredicate断定型改，传入两个参数，返回值布尔类型。一般用于方法引用，(a,b)->{a.equals(b)} 简化为方法引用A::equals;默认使用a中的方法来接受b参数

BiFunction第一个参数为被调用类，第二个参数为被调用类接受参数，第三个为返回值

方法引用：

当要传递给lambda表达式的操作，已经有实现方法了，可以使用方法引用

对象方法引用，

对象：：方法

类：：静态方法

类：：实例方法

Stream API

流操作，产生新的数据流，原来的数据不发生改变

Stream步骤：

产生一个流：一个数据源，获取一个流

中间链式（流水线式）操作：对数据源的处理

终止操作：最后才会执行中间操作链，产生结果

注意：Stream操作是延时操作，只有最后需要结果的时候才会执行

Stream和Collection集合的区别：

Stream不改变数据源的本体，只对数据操作

创建Stream的几种方法：

1

Collection接口里的方法：stream();获取串行流 parallelStream();获取并行流

2

通过Arrays中的方法stream();传入数组参数

3

Stream.iterate()方法

Stream.generate()

中间操作：

filter()过滤方法

limit()限制输出个数

skip([指定个数])跳过，跳过[指定个数]符合条件的值

distinct()去重，需要重写数据对象的equals方法和hashCode方法

map()映射方法，接受集合，返回处理后的集合

sorted()排序方法，无参数，按内部比较器比较，实现Comparable接口的CompareTo方法

sorted()排序，方法内传参数为函数接口定义比较规则

终止操作：

forEach();遍历拿到的数据，没有这个方法，中间操作不会执行；我们叫做“延迟加载”、“惰性求值”

allMatch();判断所有集合是否都符合函数式接口参数定义的条件

anyMatch();判断是否存在有符合条件的

noneMatch();判断是否没有匹配元素

findFirst();返回第一个

findAny();返回其中任意一个

count();返回流中元素数

max();实现函数接口比较方法，返回最大的值

reduce([起始数]，[函数接口]);归约方法，以第一个参数为起始数，第二个参数为运算方法，将所有数组的数依次0计算，例reduce(0,(x,y)->x+y);x是0，y是数组第一个数等到结果，x是计算结果，y是第二个数，以此类推等到最终结果

collect()；收集方法

链式操作[方法].[方法].[方法]

Stream进行迭代是内部迭代，我们以前使用的是外部迭代

Optional：

1.8新特性，解决空指针异常类的类