1. lambda的本质：作为函数式接口的实例，接口中只包含一个抽象方法称为函数式接口，在接口上可以使用@FunctionalInterface声明为函数式接口；

2. ->：lambda操作符； ->左边：接口中的抽象方法的形参列表； ->右边：重新抽象方法的方法体。

3.lambda表达式的使用：

①无参，无返回值，空小括号：

Runnable r1 = () -> System.***out***.println(**"runnable lambda形式"**);  
r1.run();

②需要一个参数但没有返回值，(String s)可改为s形式:

Consumer<String> consumer = s -> {System.***out***.println(s);};  
consumer.accept(**"lambda"**);

③需要两个或以上的参数，当方法体只有一条语句时，return与大括号都可以省略，存在return关键字，大括号{}不能省，注意语句分号：

Comparator<Integer> c1= (o1, o2) -> {**return** Integer.*compare*(o1, o2);};  
 System.***out***.println(comparator1.compare(13, 12));

4.方法引用使用格式：类(或对象) :: 方法名，方法名不需要设置参数和小括号；

5.方法引用使用的三种情况：

①对象实例名：：非静态方法；

②类 ：：静态方法；

1. 类 ：：非静态方法；

6. Stream：对集合或数组当中数据进行计算。不会存储元素；不会改变源对象，可赋值到新的stream；操作是延迟进行的，等到需要的结果是才算结束。

7. List的stream开始：stuList.stream()串行流；stuList.parallelStream()并行流；

8. Array的stream开始：Arrays.stream(new int[]{1, 2, 3});

9.List的stream中间操作方法：

①filter：对集合进行过滤操作，排除某些元素；

studentList.stream().filter(student -> student.getAge() > 13).forEach(student -> {  
 System.out.println(student);  
 });

②limit：截断流，使其元素不超过给定数量；

studentList.stream().filter(student -> student.getAge() > 10).limit(1).forEach(student -> {  
 System.out.println(student);  
 });

③skip：跳过指定个数元素；

studentList.stream().skip(1).forEach(student -> {  
 System.out.println(student);  
 });

④distinct：去重，如果是对象集合需要在对象中重写hashCode()和equals()方法；

IntList.stream().distinct().forEach(intContent -> {  
 System.out.println(intContent);  
 });

⑤map：映射，将元素转换为其他形式扩展；

strList.stream().map(content -> content.toUpperCase()).forEach(str -> {  
 System.out.println(str);  
 });

⑥sorted：自然排序，针对包装类型从小到大顺序展示；

list.stream().sorted().forEach(content -> {  
 System.out.println(content);  
 });

sorted：定制排序，针对对象按规则进行排序

studentList.stream().sorted((o1, o2) -> o1.getAge() - o2.getAge()).forEach(student -> {  
 System.out.println(student);  
 });

10.List的stream终止操作方法：

①allMatch：判断所有元素是否匹配给定的条件；

boolean result = studentList.stream().allMatch(student -> student.getAge() > 12);  
System.out.println(result);

②anyMatch：判断是否存在至少有一个匹配给定的条件；

boolean result2 = studentList.stream().anyMatch(student -> student.getAge() > 11);  
System.out.println(result2);

③noneMatch：判断是否存在没有给定条件的数据；

boolean result3 = studentList.stream().noneMatch(student -> student.getAge() > 16);  
System.out.println(result3);

④findFirst：返回第一个元素；

Optional<Student> first = studentList.stream().findFirst();  
System.out.println(first.get());

⑤findAny：返回任意一个元素；

Optional<Student> any = studentList.stream().findAny();  
System.out.println(any.get());

⑥count: 返回流中元素的总个数；

long count = studentList.stream().count();  
System.out.println(count);

⑦max：返回流中的最大值数据；

Optional<Student> max = studentList.stream().max((o1, o2) -> o1.getAge() - o2.getAge());  
System.out.println(max.get());

⑧min：返回流中的最小值数据；

Optional<Student> min = studentList.stream().min((o1, o2) -> o1.getAge() - o2.getAge());  
System.out.println(min.get());

⑨forEach：内部迭代；

studentList.stream().forEach(student -> {  
 System.out.println(student);  
});

集合自带的遍历方法：

studentList.forEach(student -> {

System.out.println(student);

});

⑩reduce：将流中的数据根据算法结合起来，求单个总值；

Integer reduce = list.stream().reduce(0, (o1, o2) -> {  
 return o1 + o2;  
 });  
 System.out.println(reduce);

⑪collect：将流内容转换为集合接受；

List<Person> collect = personList.stream().filter(student -> student.getAge() > 20).collect(Collectors.toList());  
 System.out.println(collect);

11. Optional：可以避免空指针异常，自带有几种方法。optional.of(T):对象必须存在，否则返回空指针；optional.ofNullable(T):对象可以为空，为空就返回空；optional.orElse(new K(“aaa”)):optional不为空就使用optional，否则就新创建一个；