1. Lambda 表达式是 Java SE 8 中一个重要的新特性。Lambda 表达式允许你通过表达式来代替功能接口。
2. 函数式编程（英语：functional programming）或称函数程序设计，又称泛函编程，是一种编程典范，它将计算机运算视为数学上的函数计算，并且避免使用程序状态以及易变对象。函数编程语言最重要的基础是λ演算（lambda calculus）。而且λ演算的函数可以接受函数当作输入（引数）和输出（传出值）。
3. 一个 Lambda 表达式具有下面这样的语法特征。它由三个部分组成：第一部分为一个括号内用逗号分隔的参数列表，参数即函数式接口里面方法的参数；第二部分为一个箭头符号：->；第三部分为方法体，可以是表达式和代码块。语法如下：



1. Lambda 表达式的几个最重要的特征：

可选的类型声明：你不用去声明参数的类型。编译器可以从参数的值来推断它是什么类型。

可选的参数周围的括号：你可以不用在括号内声明单个参数。但是对于很多参数的情况，括号是必需的。

可选的大括号：如果表达式体里面只有一个语句，那么你不必用大括号括起来。

可选的返回关键字：如果表达式体只有单个表达式用于值的返回，那么编译器会自动完成这一步。若要指示表达式来返回某个值，则需要使用大括号。

1. 注意点：
   1. Lambda 表达式优先用于定义功能接口在行内的实现，即单个方法只有一个接口。在上面的例子中，我们用了多个类型的 Lambda 表达式来定义 MathOperation 接口的操作方法。然后我们定义了 GreetingService 的 sayMessage 的实现。
   2. Lambda 表达式让匿名类不再需要，这为 Java 增添了简洁但实用的函数式编程能力。
2. Lambda 的作用域
   1. 可访问 static 修饰的成员变量，如果是 final static 修饰，不可再次赋值，只有 static 修饰可再次赋值；
   2. 可访问表达式外层的 final 局部变量（不用声明为 final，隐性具有 final 语义），不可再次赋值。
3. 方法引用提供了一个很有用的语义来直接访问类或者实例的已经存在的方法或者构造方法。

方法引用可以通过方法的名字来引用其本身。方法引用是通过::符号（双冒号）来描述的。

它可以用来引用下列类型的方法：

构造器引用。语法是 Class::new，或者更一般的 Class< T >::new，要求构造器方法是没有参数；

静态方法引用。语法是 Class::static\_method。

特定类的任意对象方法引用。它的语法是 Class::method。

特定对象的方法引用，它的语法是 instance::method。

1. 函数式接口是只包含一个方法的接口。
2. Predicate 是 Java 提供的一个函数式接口，他接受一个参数 t，并执行断言操作返回一个 boolean。
3. Stream 是 Java 8 开始的一个新的抽象层。通过使用 Stream，你能以类似于 SQL 语句的声明式方式处理数据。Stream 代表了来自某个源的对象的序列，这些序列支持聚集操作。
4. Stream的特性：
   1. 元素序列：Stream 以序列的形式提供了特定类型的元素的集合。根据需求，它可以获得和计算元素，但不会储存任何元素。
   2. 源：Stream 可以将集合、数组和 I/O 资源作为输入源。
   3. 聚集操作：Stream 支持诸如filter、map、limit、reduce等的聚集操作。
   4. 流水技术：许多 Stream 操作返回了流本身，故它们的返回值可以以流水的行式存在。这些操作称之为中间操作，并且它们的功能就是负责输入、处理和向目标输出。collect()方法是一个终结操作，通常存在于流水线操作的末端，来标记流的结束。
   5. 自动迭代：Stream 的操作可以基于已提供的源元素进行内部的迭代，而集合则需要显式的迭代。
5. FlatMap 用于将多个流合并为一个流，使用 FlatMap 时，表达式的返回值必须是 Stream 类型。而 Map 用于将一种流转化为另外一个流。