**Java Lambda 表达式**

## 说明：

Lambda 表达式，也可称为闭包，它是推动 Java 8 发布的最重要新特性。

Lambda 允许把函数作为一个方法的参数（函数作为参数传递进方法中）。

使用 Lambda 表达式可以使代码变的更加简洁紧凑。

## 语法：

lambda 表达式的语法格式如下：

(parameters) -> expression

或

(parameters) ->{ statements; }

虽然使用 Lambda 表达式可以对某些接口进行简单的实现，但并不是所有的接口都可以使用 Lambda 表达式来实现。***Lambda 规定接口中只能有一个需要被实现的方法，不是规定接口中只能有一个方法。***

jdk 8 中有另一个新特性：default， 被 default 修饰的方法会有默认实现，不是必须被实现的方法，所以不影响 Lambda 表达式的使用。

**@FunctionalInterface**[**#**](https://www.cnblogs.com/haixiang/p/11029639.html#3749303145)

修饰函数式接口的，要求接口中的抽象方法只有一个。 这个注解往往会和 lambda 表达式一起出现。

## 以下是lambda表达式的重要特征:

可选类型声明：不需要声明参数类型，编译器可以统一识别参数值。

可选的参数圆括号：一个参数无需定义圆括号，但多个参数需要定义圆括号。

可选的大括号：如果主体包含了一个语句，就不需要使用大括号。

可选的返回关键字：如果主体只有一个表达式返回值则编译器会自动返回值，大括号需要指定明表达式返回了一个数值。

## Lambda 表达式实例

Lambda 表达式的简单例子:

// 1. 不需要参数,返回值为 5

() -> 5

// 2. 接收一个参数(数字类型),返回其2倍的值

x -> 2 \* x

// 3. 接受2个参数(数字),并返回他们的差值

(x, y) -> x – y

// 4. 接收2个int型整数,返回他们的和

(int x, int y) -> x + y

// 5. 接受一个 string 对象,并在控制台打印,不返回任何值(看起来像是返回void)

(String s) -> System.out.print(s)

## Java示例代码：

详情见Lambda.java文件：



## 使用 Lambda 表达式需要注意以下两点：

Lambda 表达式主要用来定义行内执行的方法类型接口，例如，一个简单方法接口。在上面例子中，我们使用各种类型的Lambda表达式来定义MathOperation接口的方法。然后我们定义了sayMessage的执行。

Lambda 表达式免去了使用匿名方法的麻烦，并且给予Java简单但是强大的函数化的编程能力。

## 变量作用域：

lambda 表达式只能引用标记了 final 的外层局部变量，这就是说不能在 lambda 内部修改定义在域外的局部变量，否则会编译错误。

在 Lambda.java 文件输入以下代码：

public class lambda {

final static String salutation = "Hello! ";

public static void main(String args[]){

GreetingService greetService =

message -> System.out.println(salutation + message);

greetService.sayMessage("Runoob");

}

interface GreetingService {

void sayMessage(String message);

}

}

执行以上脚本，输出结果为：Hello! Runoob

我们也可以直接在 lambda 表达式中访问外层的局部变量：

public class lambda {

public static void main(String args[]) {

final int num = 1;

Converter<Integer, String> s = (param) -> System.out.println(String.valueOf(param + num)); s.convert(2); // 输出结果为 3

}

public interface Converter<T1, T2> {

void convert(int i);

}

}

lambda 表达式的局部变量可以不用声明为 final，但是必须不可被后面的代码修改（即隐性的具有 final 的语义）

int num = 1;

Converter<Integer, String> s = (param) -> System.out.println(String.valueOf(param + num));

s.convert(2);

num = 5;

//报错信息：Local variable num defined in an enclosing scope must be final or effectively final

在 Lambda 表达式当中不允许声明一个与局部变量同名的参数或者局部变量。

String first = "";

Comparator<String> comparator = (first, second) -> Integer.compare(first.length(), second.length()); //编译会出错

**Lambda 表达式常用示例**

## 语法：​方法归属者::方法名 静态方法的归属者为类名，普通方法归属者为对象

## lambda 表达式引用方法

有时候我们不是必须要自己重写某个匿名内部类的方法，我们可以可以利用 lambda表达式的接口快速指向一个已经被实现的方法。

示例：



## 构造方法的引用

一般我们需要声明接口，该接口作为对象的生成器，通过 类名::new 的方式来实例化对象，然后调用方法返回对象。

示例：



## lambda 表达式常用方法

## Lambda 表达式中的闭包问题

这个问题我们在匿名内部类中也会存在，如果我们把注释放开会报错，告诉我 num 值是 final 不能被改变。这里我们虽然没有标识 num 类型为 final，但是在编译期间虚拟机会帮我们加上 final 修饰关键字。

**import** java.util.function.Consumer;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**int** num = 10;

Consumer<String> consumer = ele -> {

System.***out***.println(num);

};

//num = num + 2;

consumer.accept("hello");

}

}