lambda表达式

Java 中的 Lambda 表达式通常使用 (argument) -> (body) 语法书写，例如：

```
(arg1, arg2...) -> { body }
(type1 arg1, type2 arg2...) -> { body }
```

以下是一些 Lambda 表达式的例子：

```
(int a, int b) -> {  return a + b; }

() -> System.out.println("Hello World");

(String s) -> { System.out.println(s); }

() -> 42

() -> { return 3.1415 };
```

一、Lambda 表达式的结构

让我们了解一下 Lambda 表达式的结构。

一个 Lambda 表达式可以有零个或多个参数

参数的类型既可以明确声明，也可以根据上下文来推断。例如：(int a)与(a)效果相同

所有参数需包含在圆括号内，参数之间用逗号相隔。例如：(a, b) 或 (int a, int b) 或 (String a, int b, float c)

空圆括号代表参数集为空。例如：() -> 42

当只有一个参数，且其类型可推导时，圆括号（）可省略。例如：a -> return a*a

Lambda 表达式的主体可包含零条或多条语句

如果 Lambda 表达式的主体只有一条语句，花括号{}可省略。匿名函数的返回类型与该主体表达式一致

如果 Lambda 表达式的主体包含一条以上语句，则表达式必须包含在花括号{}中（形成代码块）。

匿名函数的返回类型与代码块的返回类型一致，若没有返回则为空

二、函数式接口

在 Java 中，Marker（标记）类型的接口是**一种没有方法或属性声明的接口**，简单地说，marker 接口是空接口。相似地，**函数式接口是只包含一个抽象方法声明的接口。**

java.lang.Runnable 就是一种函数式接口，在 Runnable 接口中只声明了一个方法 void run()，相似地，ActionListener 接口也是一种函数式接口，我们使用匿名内部类来实例化函数式接口的对象，有了 Lambda 表达式，这一方式可以得到简化。

每个 Lambda 表达式都能隐式地赋值给函数式接口，例如，我们可以通过 Lambda 表达式创建 Runnable 接口的引用。

```
Runnable r = () -> System.out.println("hello world");
```

当不指明函数式接口时，编译器会自动解释这种转化：

```
new Thread(
   () -> System.out.println("hello world")
).start();
```

因此，在上面的代码中，编译器会自动推断：根据线程类的构造函数签名 public Thread(Runnable r) { }，将该 Lambda 表达式赋给 Runnable 接口。

以下是一些 Lambda 表达式及其函数式接口：

```
Consumer<Integer>  c = (int x) -> { System.out.println(x) };

BiConsumer<Integer, String> b = (Integer x, String y) -> System.out.println
(x + " : " + y);

Predicate<String> p = (String s) -> { s == null };
```

三、@FunctionalInterface注解

@FunctionalInterface 是 Java 8 新加入的一种接口，用于指明该接口类型声明是根据 Java 语言规范定义的函数式接口。Java 8 还声明了一些 Lambda 表达式可以使用的函数式接口，**当你注释的接口不是有效的函数式接口时，可以使用 @FunctionalInterface 解决编译层面的错误**

```
 //定义一个函数式接口
 //用了该注解的的接口只能拥有一个方法
@FunctionalInterface
public interface WorkerInterface {
```

```
    public void doSomeWork();

}


public class WorkerInterfaceTest {

    public static void execute(WorkerInterface worker) {
        worker.doSomeWork();
    }

    public static void main(String [] args) {

        //invoke doSomeWork using Annonymous class
        execute(new WorkerInterface() {
            @Override
            public void doSomeWork() {
            System.out.println("Worker invoked using Anonymous class");
            }
        });

    //invoke doSomeWork using Lambda expression
    execute( () -> System.out.println("Worker invoked using Lambda expressi
on") );
    }

}
```