链滴

Java8:Lambda 入门教程

作者：san
原文链接：https://ld246.com/article/1533612385296
来源网站：链滴

Lambda简介

Lambda表达式本质上是匿名方法，其底层还是通过invokedynamic指令来生成匿名类来实现。它提了更为简单的语法和写作方式，允许你通过表达式来代替函数式接口。

Lambda表达式，可以让我们的代码变得简洁，并且可以通过数据流的方式处理集合。

函数式接口的概念

因为Lambda表达式应用了函数接口，我们先了解一下函数式接口的概念。

函数式接口即只有一个抽象方法的接口。函数式接口可通过注解@FunctionalInterface定义。

Java8中内置了四大核心函数式接口：

```
1)消费型接口：
interface Consumer{
    void accept(T t);
}

2)供给型接口：
interface Supplier{
    T get();
}

3)函数型接口：
interface Function{
    R apply(T t);
}

4)断言型接口：
interface Predicate{
    boolean test(T t);
}
```

我们可以根据参数的个数以及返回值类型，选用合适的函数式接口。

Lambda基本语法

```
(parameters) -> expression
或
(parameters) ->{ statements; }
```

parameters：参数，这里的参数指的是函数接口里的参数。可明确表示，也可以由JVM进行腿短。

->：指的是参数用于右边的expression和statements。（方法体）

expression或statements：可以理解为方法体

下面通过几个示例看一看如何使用:

```
//示例1：不接受参数，直接返回5
```

```
() -> 5
//示例2：接受两个String类型参数，并打印出来
(String x, String y) -> System.out.println( x + y);
//示例3：接受x，y两个参数，参数类型由JVM根据上下文推断出来，并返回两个参数的和
(x, y) -> x+y;
```

Lambda的接口使用

Lambda表达式的目标类型是函数性接口——每一个Lambda都能通过一个特定的函数式接口与一个定的类型进行匹配。

自定义函数接口使用示例

```
@Functional
Interface interface Converter{
  T convert(F from);
}
```

使用传统方式使用该接口：

```
Converter converter=new Converter() {
@Override
public Integer convert(String from) {
  return Integer.valueOf(from); }
};
Integer result = converter.convert("200");
System.out.println(result);
```

但是如果使用Lambda表达式的话，代码就可以变的很简洁：

```
Converter converter=(param) -> Integer.valueOf(param); //将overrid写在表达式中
Integer result = converter.convert("101");
System.out.println(result);
```

实现Runnable接口

1.1使用匿名内部类

```
new Thread(new Runnable() {
    @Override
    public void run() {
        System.out.println("Hello world !");
    }
}).start();
```

1.2使用 lambda expression

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

2.1使用匿名内部类

```
Runnable race1 = new Runnable() {
    @Override
    public void run() {
        System.out.println("Hello world !");
    }
};
```

2.2使用 lambda expression

```
Runnable race2 = () -> System.out.println("Hello world !");

// 直接调用 run 方法(没开新线程)
race1.run();
race2.run();
```

Lambda循环使用

```
String[] names={"Mike","Mary","Eric","Lucy","Rose"};
List<String> students=Arrays.asList(names);
// 以前的循环方式
for (String student: students) {
     System.out.print(student+ "; ");
}

//方式一： 使用 lambda 表达式以及函数操作
students.forEach((student) -> System.out.print(student+ "; "));

// 方式二：使用双冒号操作符（Lambda表达式方法引用的第一种语法格式，下面会讲 ）
students.forEach(System.out::println);
```

方法引用

方法引用，有三种语法格式：

1)对象::实例方法名

例：Consumer con=(x)->System.out.println(x);可以写为：

Consumer con=System.out::println;//System.out即PrintWriter对象

2)类::静态方法名

例：Comparator com=(x,y)->Integer.compare(x,y);可以写为：

Comparator com=Integer::compare;

3)类::实例方法名

例：BiPredicate bp=(x,y)->x.equals(y);

BiPredicate bp=String::equals;

静态方法使用

传统情况下，我们可能写出这种代码：

```
public class User {
    public static void main(String[] args) {
        Converter<String ,Integer> converter=new Converter<String, Integer>() {
            @Override
            public Integer convert(String from) {
                return User.String2Int(from);
            }
        };
        converter.convert("120");
    }


    @FunctionalInterface
    interface Converter<F,T>{
        T convert(F from);
    }

    static int String2Int(String from) {
        return Integer.valueOf(from);
    }
}
```

如果使用Lambda表达式，静态方法引用的话：

```
public class User {

  public static void main(String[] args) {
    Convert<String, Integer> convert = User::string2Int;
    System.out.println(convert.convert("120"));

  }

  @FunctionalInterface
  interface Convert<F, T> {
    T convert(F from);
  }

  static int string2Int(String from) {
    return Integer.valueOf(from);
  }
}
```

将Override改为一句话：

Converter converter = User::String2Int;

使用User类里的静态方法string2Int。

实例方法使用*

传统代码中：

```
public class User {
    public static void main(String[] args) {

        Converter<String, Integer> converter = new Converter<String, Integer>() {
            @Override
            public Integer convert(String from) {
                return new Helper().String2Int(from);
            }
        };
        converter.convert("120");
    }


    @FunctionalInterface
    interface Converter<F, T> {
        T convert(F from);
    }

    static class Helper {
        public int String2Int(String from) {
            return Integer.valueOf(from);
        }
    }
}
```

使用Lambda表达式中的方法引用的话，十分简洁：

```
Helper helper = new Helper();
Converter<String, Integer> converter = helper::String2Int;
converter.convert("120");
```

Stream方面的应用，因为目前没有怎么看流，所以准备等熟悉了解流之后再完整笔记。