函数式编程

函数式编程是一种“编程范式”（Programming paradigm），如何写程序的方法论，其是属于结构化编程的一种，主要思想是把运算过程尽量写成一系列嵌套的函数调用，其特点：

* 函数是第一等公民（First Class），函数和其他数据类型一样，处于平等地位，可以赋值给其他变量，也可以作为参数传入另一个函数，或者作为别的函数返回值
* 只用表达式，不用语句，“Expression”是一个单纯的运算过程，总是有返回值；语句（Statement）是执行某种操作，没有返回值。函数式编程要求，只使用表达式，不使用语句，也就是说每一步都是单纯的运算，而且都有返回值
* 没有副作用，副作用是指函数内部与外部互动产生运算以外的其他结果。函数式编程强调没有副作用，即函数要保持独立，所有功能就是返回一个新值，没有其他行为，尤其是不得修改外部变量值
* 不修改状态，只返回新的值，不修改系统变量。在函数式编程中使用参数保持状态，最好的例子是递归
* 引用透明，函数的运行不依赖外部变量或者状态，只依赖于输入的参数，任何时候只要参数相同，引用函数所得到的返回值总是相同的。

使用函数式编程使代码简洁、开发快速、接近自然语言，易于理解，在函数编程中不需要考虑死锁等易于并发编程。函数式编程没有副作用，只要保证接口不变，内部实现和外部无关，因此代码管理更方便，可以在运行状态下直接升级代码。

# **Lambda表达式语法**

Lambda是Java SE 8中的一个重要特性，其本质只是一个”语法糖”，由编译器推断并转换包装为常规的代码。Lambda表达式是一个匿名函数，允许通过表达式替代功能接口，在表达式中提供正常的参数列表和使用参数的body（代码块），Lambda的标准语法如下：

*(Type1 param1, Type2 param2, …, TypeN paramN) -> {  
statment1;  
statment2;  
// …….  
stamentN;  
return something;*

*}*

上述的语法有多种简化版本：

1. 省略参数类型，编译器可以从上下文环境中推断出Lambda表达式的参数类型，其表达式为：

*(param1, param2, …, paramN) -> {  
statment1;  
statment2;  
// …….  
stamentN;  
return something;*

*}*

1. 当参数个数只有一个，可以省略参数的小括号，简写为：

*param1-> {  
statment1;  
statment2;  
// …….  
stamentN;  
return something;*

*}*

1. 当表达式只包含一条语句时，可省略大括号、return和语句结尾的分好，简化为：

*param1-> statement*

1. 示例

*() -> 5*

*x -> 2\*x*

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

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

*(String s) -> System.out.println(s)*

执行示例：

*inventory.sort((Apple a1, Apple a2) -> a1.getWeight().compareTo(a2.getWeight()));*

https://blog.csdn.net/chuanben/article/details/52346129

https://blog.csdn.net/IO\_Field/article/details/54380200

# **函数式接口**

下面是Lambda表示的使用示例，定义方法如下：

*public static List<Apple> filterApples(List<Apple> inventory, Predicate<Apple> p) {*

*List<Apple> result = new ArrayList<>();*

*for(Apple apple: inventory) {*

*if(p.test(apple)) {*

*result.add(apple);*

*}*

*}*

*return result;*

*}*

其中Predicate接口如下：

*public interface Predicate<T> {*

*boolean test(T t);*

*}*

定义方法的调用示例：

*List<Apple> redApples = filterApples(inventory,*

*(Apple apple) -> "red".equals(apple.getColor())*

*);*

在示例中将Lambda表达式(标红部分)传给filterApples方法，之所以可以这么用是因为为了参数化filterApps方法的行为而创建了Predicate<T>接口，在接口中仅定义了一个抽象方法。 这就引入了函数式接口，其为定义一个抽象方法的接口。Lambda表达式允许直接以内联的形式为函数式接口的抽象方法提供实现，并把整个表达式作为函数式接口的实例（具体来说，是函数式接口一个具体实现的实例）。

函数式接口的抽象方法的Signature基本上是Lambda表达式的签名，这种抽象方法称为函数描述符。Lambda和函数式接口的签名可以使用下面的表示法：

*() ->void //参数类表为空，且返回void函数*

*(Apple,Appline) -> int，代表接收两个Apple为参数且返回int的函数*

Java8为函数式接口引入一个新注解@FunctionalInterface，用于编译级错误检查，当接口不符合函数式接口的定义时，编译器会报错。函数式接口的定义要求：

* 只有一个抽象方法
* 允许包含默认方法，不是抽象方法，其有一个默认实现
* 允许定义静态方法
* 允许定义java.lang.Object中的public方法

函数式接口的定义及测试如下：

*@FunctionalInterface*

*public interface Converter<F,T> {*

*T convert(F from);*

*}*

*//测试调用*

*Converter<String,Integer> converter = (from) -> Integer.parseInt(from); //实现convert方法*

*Integer integer = converter.convert("231");*

Lambda表达式可以理解为对函数式接口和其中抽象方法的具体实现，当需要一个函数式接口参数的方法是，可以传递一个对应的Lamba表达式作为参数。

在JDK中自带的函数式接口：Runnable、Comparator、Callable及java.util.function包下的接口，其中内置的4大核心函数式接口：

1. Consumer<T>，消费型接口，其定义如下：

*@FunctionalInterface  
public interface Consumer<T> {  
 //执行操作，输入参数为T，返回空值  
 void accept(T t);  
 //构成执行序列  
 default Consumer<T> andThen(Consumer<? super T> after) {  
 Objects.requireNonNull(after);  
 return (T t) -> { accept(t); after.accept(t); };  
 }  
}*

用于对类型为T的对象应用操作，包含的方法:void accept(T t)

1. Supplier<T>，供给型接口，其定义如下：

*@FunctionalInterface  
public interface Supplier<T> {  
 T get();  
}*

返回类型为T的对象，包含方法：T get()

3）Function<T,R>，函数型接口

*@FunctionalInterface*

*public interface Function<T, R> {*

*R apply(T t);*

*//构成组合方法*

*default <V> Function<V, R> compose(Function<? super V, ? extends T> before) {*

*Objects.requireNonNull(before);*

*return (V v) -> apply(before.apply(v));*

*}*

*//构成方法执行链*

*default <V> Function<T, V> andThen(Function<? super R, ? extends V> after) {*

*Objects.requireNonNull(after);*

*return (T t) -> after.apply(apply(t));*

*}*

*//返回参数*

*static <T> Function<T, T> identity() {*

*return t -> t;*

*}*

*}*

对类型为T的对象应用操作，并返回类型为R类型的对象，包含方法: R apply(T t)

4）Predicate<T>，断言性接口

*@FunctionalInterface  
public interface Predicate<T> {  
 //返回断言  
 boolean test(T t);*

*}*

确定类型为T的对象是否满足约束，并返回boolean值，包含方法：boolean test(T t)

在Java中还提供了很多其他接口，如下所示：

