链滴

Java 8 函数式编程接口

作者：lonelyant

原文链接：https://ld246.com/article/1555570382933

来源网站：链滴

最近看了Java 8的一些新特性，作为码农到现在都没有系统了解过Java 8实属惭愧。不过亡羊补牢为不晚，现在就把学习中的知识点都记录下来。先从函数式编程开始。

以下都是最基本的用法

函数式编程/lambda

接口	输入参数	返回类型	说明
Predicate<T>	T	boolean	断言
Consumer<T>	T	/	消费一个数据
Function<T,R>	T	R	输入T输出R的函数
Supplier<T>	/	T	提供一个数据
UnaryOperator<T>	T	T	一元函数（输出输入类型相同）
BiFunction<T, U, R>	(T, U)	R	2个输入的函数
BinaryOperator<T>	(T,T)	T	二元函数（输出输入类型相同）

开始之前，先定义一个基本类，方便后面演示

```
class Dog {

    private String name = "哮天犬";

    private int food = 10;

    public Dog() {

    }

    public Dog(String name) {
        this.name = name;
    }

    public static void bark(Dog dog) {
        System.out.println(dog + "叫了");
    }

    public int eat(int num) {
        System.out.println("吃了" + num + "斤狗粮");
        this.food -= num;
        return this.food;
    }
```

原文链接：Java 8 函数式编程接口

```
    @Override
    public String toString() {
        return this.name;
    }
}
```

Predicate<T> 接口

断言函数，输入T，返回Boolean。

```
Predicate<Integer> predicate1 = i -> i < 10;
//IntPredicate predicate1 = i -> i < 10;// 等效写法
System.out.println(predicate1.test(11));// false
Predicate<Integer> predicate2 = i -> i > 8;
System.out.println(predicate1.and(predicate2).test(9));// true

Predicate<String> predicate3 = i -> i.equals("666");
System.out.println(predicate3.test("666"));// true
```

Consumer<T>接口

消费函数，输入T，没有输出，典型的消费者类型

先来看一个最简单的例子

```
Consumer<String> stringConsumer = System.out::println;
stringConsumer.accept("Hello World");    // Hello World
```

上面这个例子就不多说了，我们来看看下面这个略微有点复杂的例子

```
List<Integer> integers = new ArrayList<>();
Consumer<Integer> integerConsumer = integer -> {
    if (integer < 10){
        integers.add(integer);
    }
};
integerConsumer = integerConsumer.andThen(integer -> {
    integers.removeIf(predicate2.negate());
});
Stream.of(1,2,3,4,5,6,7,8,9,0,10).forEach(integerConsumer);
System.out.println(integers); // [9]
```

这个例子中使用了Consumer<T>接口中定义的predicate2断言函数取，使用消费者函数筛选出了8-10之间的数。

Function<T,R>接口

输入T，输出R

```
Function<Integer, Integer> times2 = i -> i*2;

Function<Integer, Integer> squared = i -> i*i;
```

```
System.out.println(times2.apply(4));

System.out.println(squared.apply(4));

//32              先4×4然后16×2,先执行apply(4)，在times2的apply(16),先执行参数，再执行调用者

System.out.println(times2.compose(squared).apply(4));

//64              先4×2,然后8×8,先执行times2的函数，在执行squared的函数。
System.out.println(times2.andThen(squared).apply(4));

//1024            先4*4，然后16*2，然后16*16
System.out.println(times2.andThen(squared).compose(squared).apply(4));

//16
System.out.println(Function.identity().compose(squared).apply(4));
```

以上例子涉及到了两个方法andThen、compose，compose为之前执行，andThen为之后。

```
Dog dog = new Dog();

IntFunction dogFunction = dog::eat;
System.out.println("还剩下" + dogFunction.apply(2) + "斤");

// 吃了2斤狗粮
// 还剩下8斤
```

Supplier<T> 接口

无输入，输出T，典型的生产者模型

```
Supplier<Dog> supplier = Dog::new;
System.out.println("创建了新对象：" + supplier.get());

// 创建了新对象：哮天犬
```

UnaryOperator<T>接口

输出、输入均为T

```
Dog dog = new Dog();

UnaryOperator<Integer> function = dog::eat;
IntUnaryOperator function2 = dog::eat;

System.out.println("还剩下" + function.apply(2) + "斤");
System.out.println("还剩下" + function2.applyAsInt(2) + "斤");

// 吃了2斤狗粮
// 还剩下8斤
// 吃了2斤狗粮
// 还剩下6斤
```

BinaryOperator<T>接口

输入为（T,T），返回为T

```
BinaryOperator<String> binaryOperator = (s1,s2) -> s1 + "====" + s2;
System.out.println(binaryOperator.apply("a","b"));

// a====b
```

BiFunction<T>接口

输入为（T,U），返回为R

```
Dog dog = new Dog();
BiFunction<Dog, Integer, Integer> eatFunction = Dog::eat;
System.out.println("还剩下" + eatFunction.apply(dog, 2) + "斤");

//吃了2斤狗粮
//还剩下8斤
```