JDK8函数式编程(一)

Tim

大纲

- JAVA8 FP 基础
  - FP 特点
  - Lambda表达式基本语法
  - 函数式接口
  - 方法引用

- JAVA8 FP 进阶
  - Stream 基础API
  - Filter/map/Reduce
  - Transducer

什么是函数式编程

- 一种编程范式
- 函数作为第一对象
- 注重描述而非具体执行步骤
- 更关心代数结构之间的关系
- 不可变

ERLANG

elixir

Haskell

OCaml

Scala

Clojure

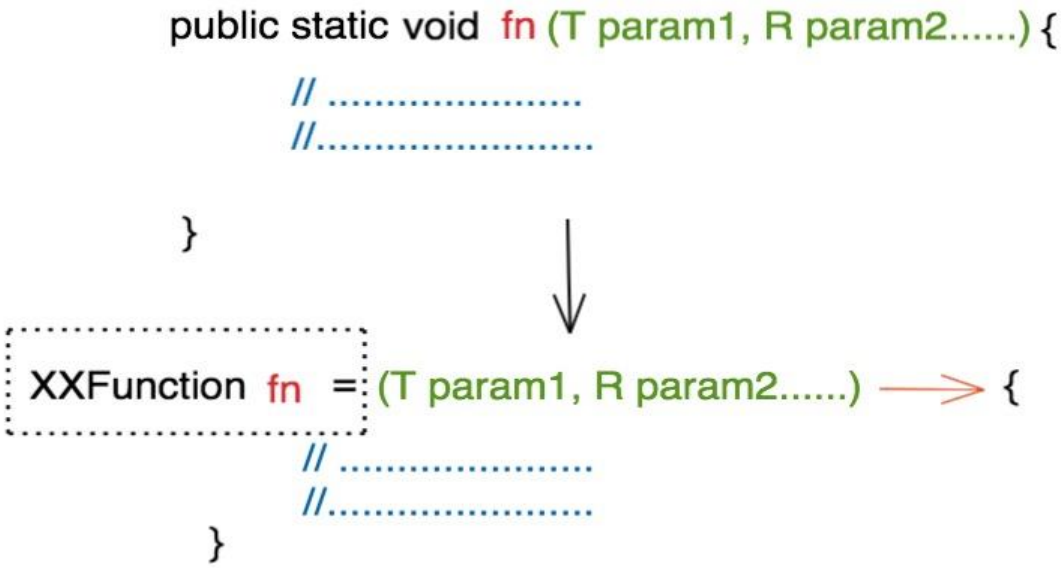
函数式编程的编程特性

- 不可变
- 惰性求值
- 闭包
- 高阶函数
- 柯里化
- 部分应用
- 结合率

JAVA 8之前早已存在的适用函数式编程的场景

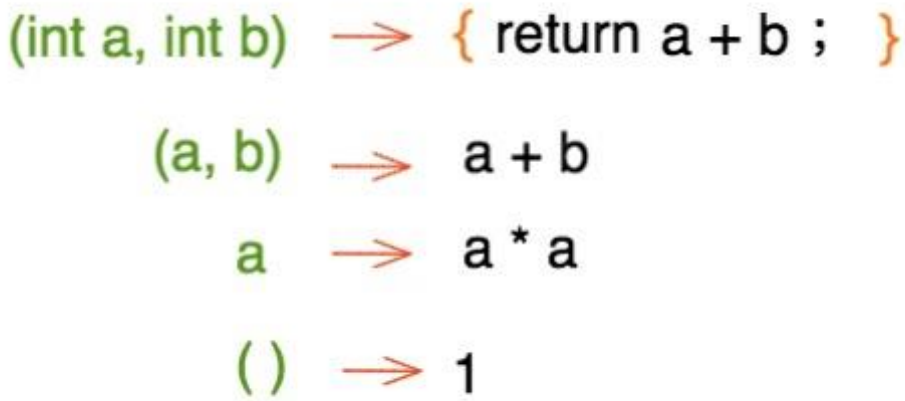
- 创建线程
- 策略模式，如comparator
- UI 编程
- 异步回调
- …

Lambda 表达式


public static void fn (T param1, R param2.....) {
// .....
//.....
}
XXFunction fn = (T param1, R param2.....) -> {
// .....
//.....
}

Lambda 表达式语法糖

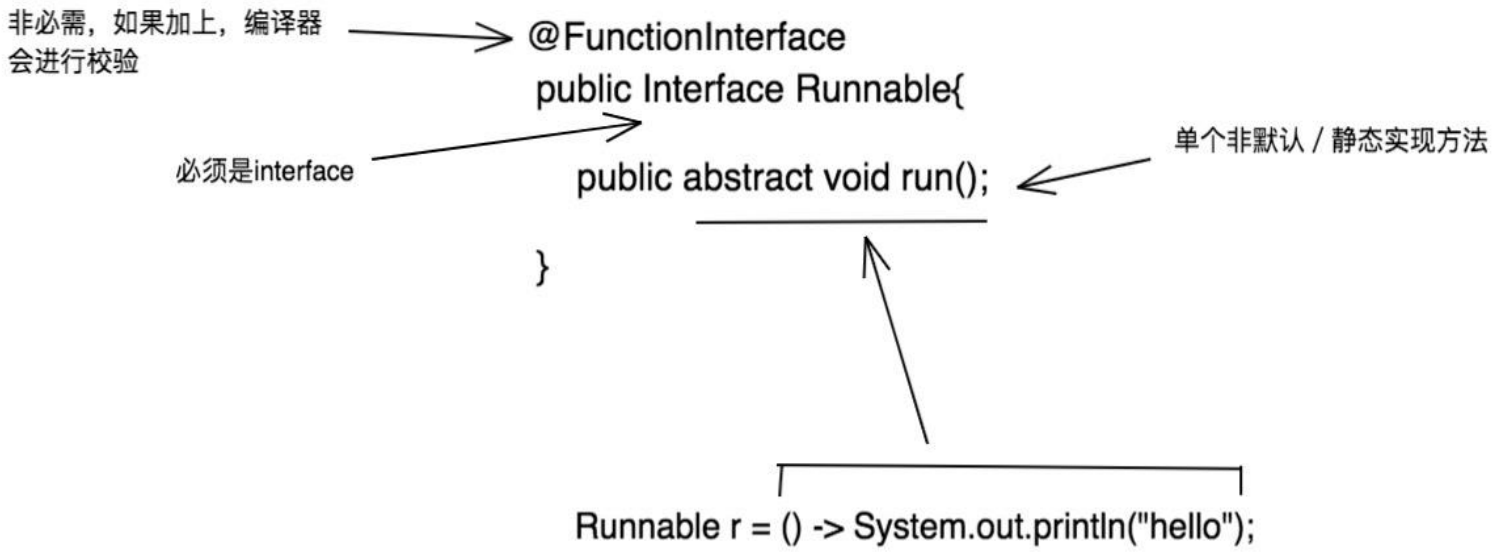
- 参数类型可推导
- 单行可省略大括号
- 单参数可省略小括号



思考：

a -> b -> c -> d 代表什么?

函数式接口 SAM（Single Abstract Method）



内置常用函数式接口

输入	返回值	class
T	R	Function<T,R>
void	T	Supplier<T>
T	void	Consumer<T>
void	void	Runnable
T	Boolean	Predicate<T>
T	T	UnaryOperator<T>

方法引用

- 静态方法 -> 需要告之属于哪个类
- 构造方法 -> 需要告之属于哪个类
- 指定实例方法 ->需要告之属于哪个实例

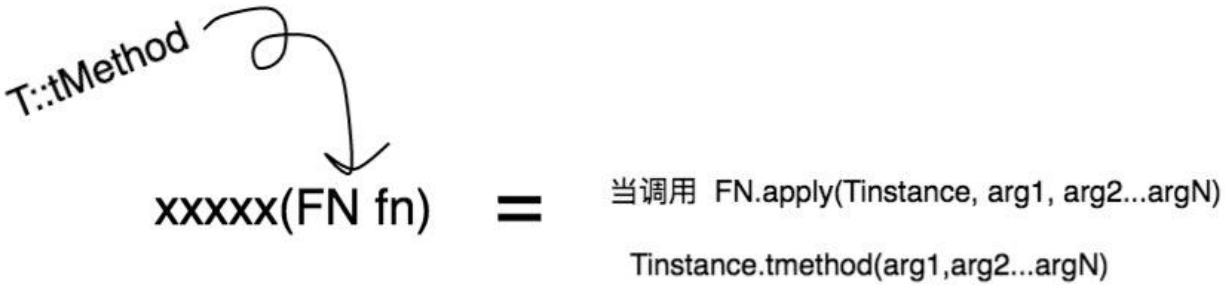
```
classX :: methodName
classX :: new
instance :: methodName
```

方法引用

指定类型任意实例方法引用

```
public class T{
    // ...
    tMethod(arg2..argN)
    // ...
}
```

```
public interface FN{
    apply(T arg1, arg2..argN)
}
```



函数式接口转换

由于JAVA是强类型，在某些场合下，我们并不要求函数签名完全一致时，可以进行转换

- 忽略输入： Function <- Supplier
- 忽略返回： Consumer <- Function
- 忽略输入和返回： Runnable <- Supplier

特殊的void-compatibility 规则：

如果lambda是一个语句表达式，那么即使该lambda有返回值也可以赋值给返回值签名为void的函数

Stream

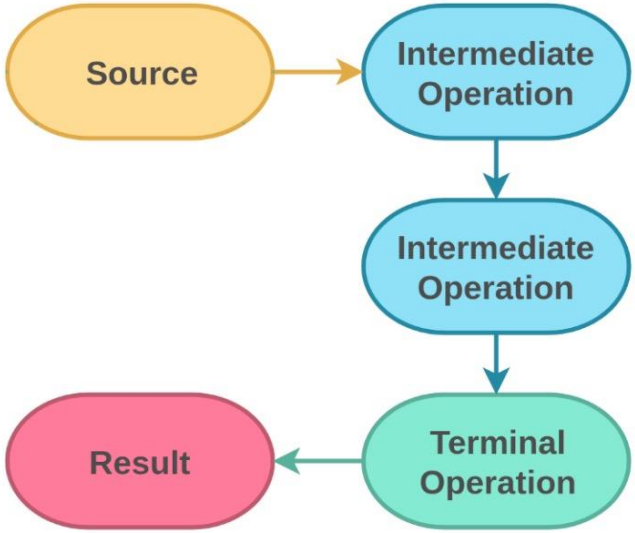
Stream VS List

- Stream 可以是无限的
- Stream可并行处理
- Stream可能延迟处理

创建Stream：

- 静态数据 Stream.of()
- 容器　　collection.stream()
- 动态　　Stream.iterate() & Stream.generate ()
- 其他api：　Files.lines()…
-

Stream 基本操作



Intermediate

- filter
- distinct
- skip
- limit
- map/flatMap
- sorted

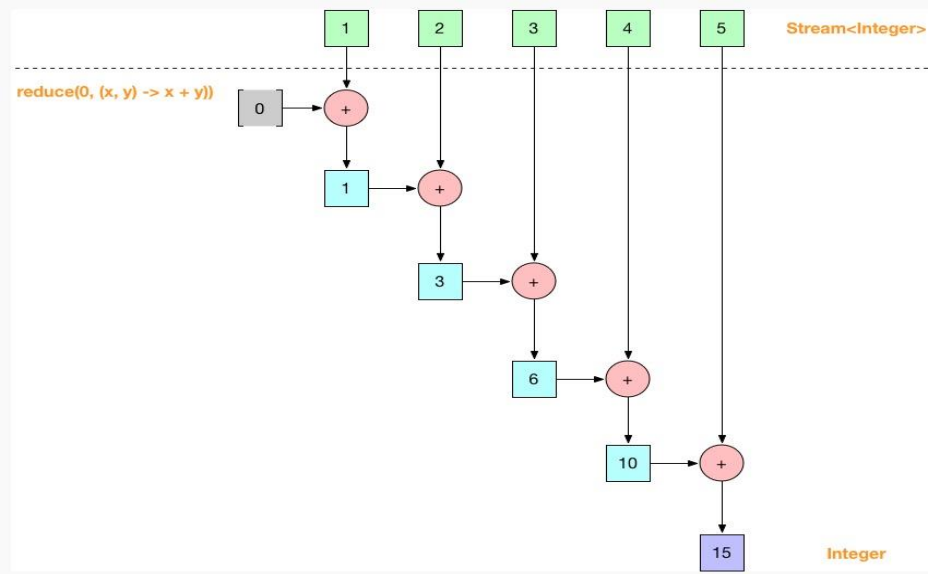
Terminal

- count/sum
- collect/reduce
- forEach
- anyMatch/allMatch/noneMath

FP 编程三板斧

- filter(Predicate predicate)
- map(Function mapper)
- reduce(U identity, BinaryOperator acc)

理解reduce




```
T reduce (T identity, BinaryOperator<T> accumulator)
                      ||
(acc, curr) -> {//... return newAcc;}
```

=

```
R ret = initValue;
for (T data : datas) {
    ret = accumulator.apply(ret, data);
}
return ret;
```

reduce 实战

- 求和
- 求最大值 / 最小值
- 串联成字符串
- 存放进collection
- 用reduce实现map
- 用reduce实现filter
- * transducer

What’s next

- Optional
- Stream.collect()
- Curry
- FunctionCompose
- Functor & Monad