## Lambda表达式

### "->"将Lambda表达式分为两部分

左侧：Lambda表达式的参数列表

右侧：Lambda表达式中所需要执行的功能，即Lambda体

### 语法格式一：无参数、无返回值

()->System.out.println("Hello Lambda"+num)

### 语法格式二：有一个参数，无返回值

(x)->System.out.println(x)

### 语法格式三：若只有一个参数，小括号可以省略不写

x->System.out.println(x)

### 语法格式四：有两个以上的参数，有返回值，并且Lambda体中有多条语句

Comparator<Integer> com = (x,y)->{

System.out.println("函数式接口");

return Integer.conpare(x,y);

}

### 语法格式五： Lambda体中只有一条语句，return和大括号都可以省略不写

Comparator<Integer> com = (x,y)-> Integer.conpare(x,y);

### 语法格式六： Lambda表达式的参数的数据类型可以省略不写，因为JVM编译器通过上下文可以推断出数据类型，即“类型推断”

(Integer x, Integer y)-> Integer.conpare(x,y);

## 函数式接口

### Lambda表达式需要“函数式接口”的支持

函数式接口：接口中只有一个抽象方法的接口，称为函数式接口

函数式接口注解：@FunctionalInterface，该注解可以检查是否式函数式接口

### Java8内置的四大核心函数式接口

Consumer<T>：消费型接口

void accept(T t);

Supplier<t>:供给型接口

T get();

Function<T,R>:函数式接口

R apply(T t)

Predicate<T>:断言型接口

boolean test(T t);

## 方法引用与构造器引用

方法引用：

若Lambda体中的内容有方法已经实现了，我可以用“方法引用”（可以理解为方法引用的是Lambda表达式的另外一种表现形式）

前提：

Lambda体中调用方法的参数列表、返回值和函数式接口的中抽象方法参数列表、返回值一定要一致

主要有三种语法格式：

对象：：实例方法名

Empty emp = new Empty();

Supplier<String> sup =() ->emp.getName()

Supplier<String> sup1 =emp::getName

String name = sup1.get()

类：：静态方法名

Comparator<Integer> com = (x,y)->Integer.compare(x,y);

Comparator<Integer> com = Integer::compare;

类：：实例方法名

BiPredicate<String,String> bp = (x,y)->x.equals(y);

BiPredicate<String,String> bp =String::equals;

boolean flag = bp.methodName(s,y);

### 构造器引用

Supplier<Employee> sup =()->new Employee();

Supplier<Employee> sup1 =()->Employee::new;

Employee emp = sup1.get();

### 数组引用（数组也是一个对象）

Function<Integer,String[]> fun =(x)->new String[];

String[] strs = fun.apply(10);

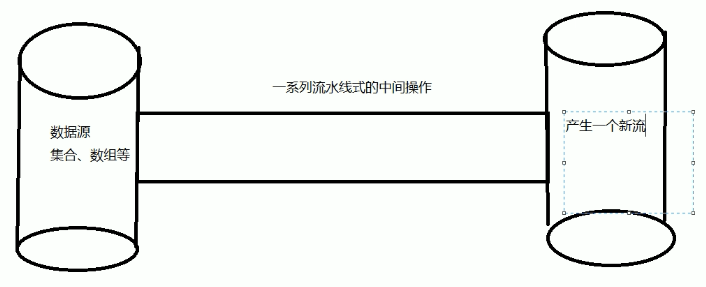
System.out.println(strs.length);

Function<Integer,String[]> fun2 =String[]::new;

String[] strs2 = fun.apply(10);

## Stream API

### 流是对数据的操作



流（Stream）是数据管道，是用于操作数据源（集合、数组）所生成的元素序列

“集合讲的是数据，流讲的是计算”

注意：

Stream自己不会储存元素

Stream不会改变源对象，相反，他会返回一个持有原结果的新Stream

Stream操作是延迟执行的。这意味着他们会等到需要结果的时候才执行

### Stream操作的三个步骤

创建Stream

一个数据源（集合、数组），获取一个流。

可以通过Collection系列集合中提供的stream()或parallelStream();

|  |
| --- |
| List<String> list = new ArrayList();  String<String> Stream = list.stream(); |

通过Arrays中的静态方法Stream获取数组流

|  |
| --- |
| Employee [] emps = new Employee[10];  Stream<Employee> Stram2 = Arrays.Stream(emps); |

通过Stream类中的静态方法of()

|  |
| --- |
| Stream<String> Stream3 = Stream.of("aa","bb","cc"); |

创建无限流

|  |
| --- |
| //迭代  Stream<Integer> Stream4 = Stream.iterate(0,(x)->x+2);  //生成  Stream.generate(()->Math.random); |

中间操作

一个中间操作链，对数据源进行处理

多个中间操作可以连接起来形成一个流水线，除非流水线上触发终止操作，否则中间操作不会进行任何处理，而再终止操作时，一次性全部处理，称为惰性求值。

筛选与切片

filter---接受Lambda，从流中排除某些元素

limit---截断流，使元素不超过给定数量

skip(n)---跳过元素，返回一个扔掉前面n个元素的流，若元素中不足n个，则返回一个空流，与limit互补

distinct----筛选，通过流所生成元素的hashCode()和equals()去除重复元素

映射

map---接受一个Lambda，将元素转换成其他形式提取信息，接受一个函数作为参数，该函数会被应用到每个元素上，并将其映射成一个新的元素。

flatMap---接受一个函数作为参数，将流中的每个值都换成另一个流，然后把所有流连接成一个流

排序

sorted()---自然排序

sorted(Comparator com)----定制排序

终止操作

一个终止操作，执行中间操作链，并产生结果

查找与匹配

allMath---检查是否匹配所有元素

anyMatch---检查是否至少匹配一个元素

noneMatch---检查是否没有匹配的元素

findFirst---返回第一个元素

findAny---返回当前流中的任意元素

count---返回流中元素的个数

max---返回流中最大的值

min---返回流中最小的值

## 类型推断升级

类型推断的例子

正确

String [] str ={"abc","adc","acd"}

错误

String [] str

str = {"abc","adc","acd"}

## HashMap数据结构

JDK1.7-------HashMap的数据结构是 数组、链表

JDK1.8-------HashMap的数据结构是 数组、链表、红黑树