Iostream

函数式接口：只有一个抽象方法的接口

Lambda表达式相当于定义了输入和输出形式，隐藏了操作，吧操作放到你需要定义的时候再定义，并且以lambda表达式的形式来表示/

四大核心函数式接口：

Consumer<T> 消费型接口： 给一个类型，无返回值

Supplier<T> 供给型接口： 不给类型，返回一个类型

Function<T,R> 函数型接口：给一个类型，返回另一个类型

Predicate<T> 断言型接口：给一个类型，返回bool值

方法引用：

对象：：方法实例名。方法的输入和返回参数需要和接口相同

类：：静态方法Integer::Compare

类：：实例方法。Lambda参数列表的第一参数是实例方法的调用者，第二个参数是实例方法的参数时，可以使用

例如X.equals(Y)等价于 String::quals

Stream自己不会存储元素，

也不会改变源对象，相反会返回一个持有结果的新Stream,

Stream操作是延迟执行的，意味着会等到需要结果的时候才执行。也就是说中间操作不会返回结果

Stream有of方法Stream.of(“we”,”df”,”asd”);

创建无限流：Stream.iterate(0, (x)->x+2);

Map的作用：将元素转换成其他形式或提取信息，接受一个函数作为参数，改函数将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，将流中的每个值都缓存另一个流，然后把所有流形成一个流

Allmatch：检查是否匹配所有元素

Anymatch：检查是否至少匹配一个元素

Nonematch：检查是否没有匹配所有元素

Find first：返回第一个元素 返回optional，然后用get获取值

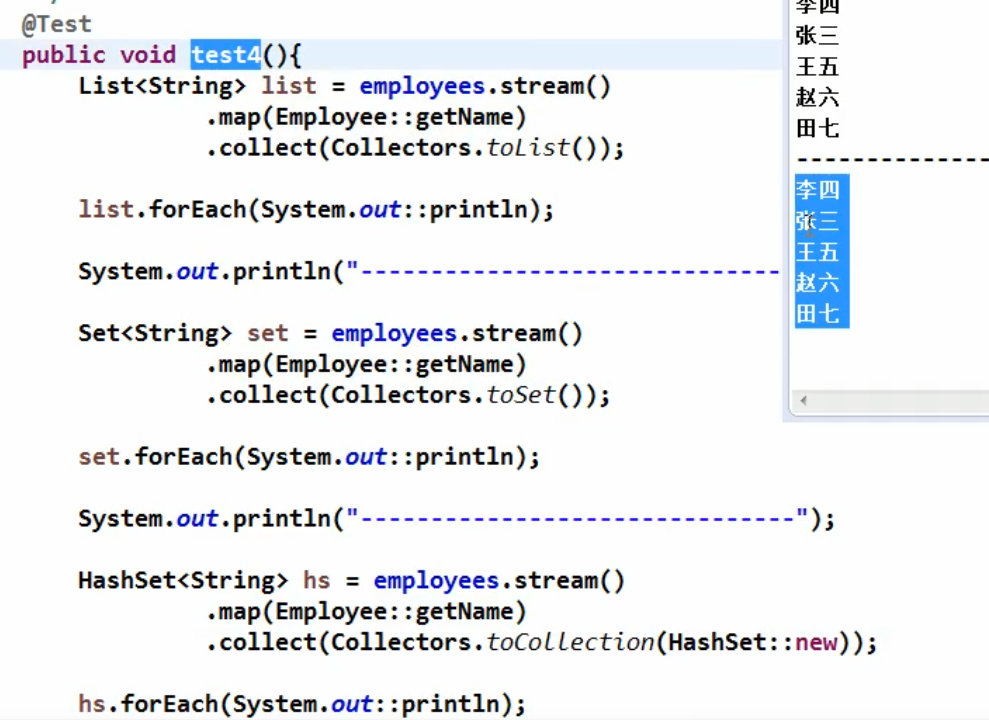
Findany：返回当前流中的任意元素

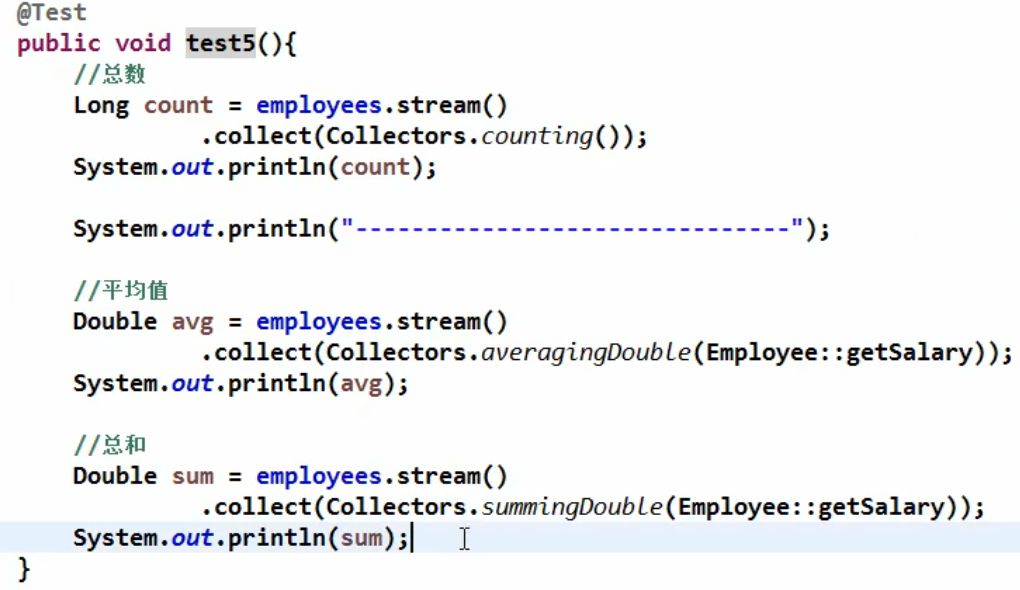
Count：返回流中元素的个数

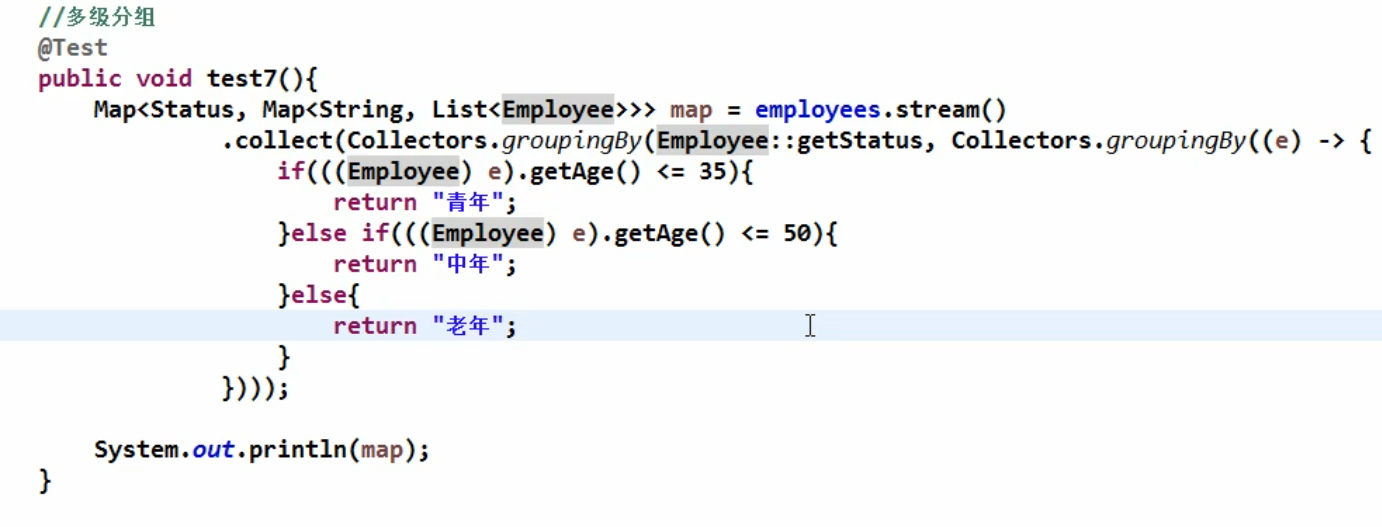
Max:返回流最大值，同理min

Reduce:归约，将流中元素反复结合起来，得到一个值，例如sum: reduce(Double::sum)

Collect：将流转化为其他形式，接受一个collector接口，用于给stream中元素做汇总的方法。

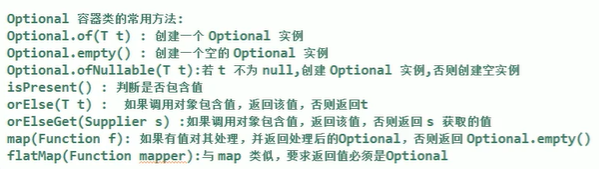






Peek:接受一个consumer<T>函数，peek会按照consumer函数提供的逻辑去消费流中的每一个元素，同时有可能改变元素内部的一些属性。

Optional:



Groupingby： .Collect(Collectors.groupingby()) function接口，例如String::length按字符串长度分，

日期API：

LocalTime lt=LocalTime.now();  
System.out.println(lt);  
  
Instant is=Instant.now();  
System.out.println(is);

DateTimeFormatter dtf =DateTimeFormatter.ISO\_DATE;  
DateTimeFormatter dtf2 =DateTimeFormatter.ofPattern("yyyy年MM月dd日 HH:mm::ss");

解析：

LocalDateTime newldt=ldt.parse(ldt.format(dtf2),dtf2);