java1.8新特性

1. **简介**

1）、速度更快。体现在对底层数据结构的优化（对hashmap、hashset、ConcurrentHashMap的优化）、对垃圾回收机制（内存结构）的调整、对于并行有更好的扩展和支持

2）、代码更少（增加了新的语法lambda表达式）

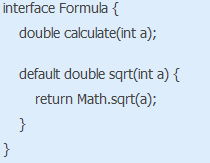
3）、强大的Stream API

4）、便于并行

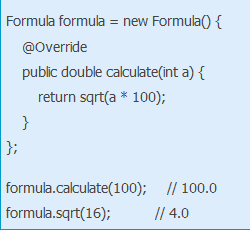
5）、最大化减少空指针异常Optional

1. **接口的默认方法**

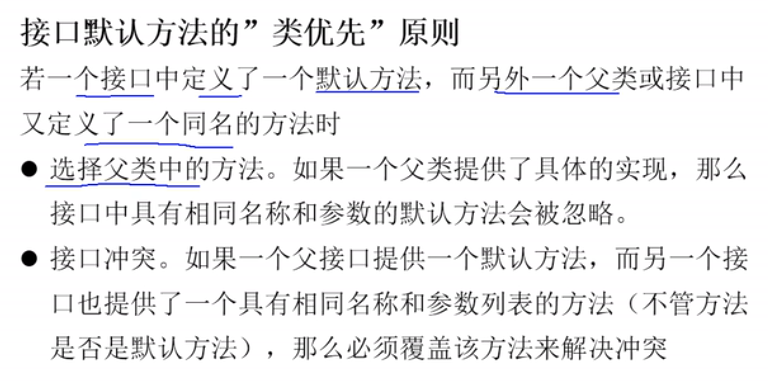
java 8允许我们给接口添加一个非抽象的方法实现，只需要使用 default关键字即可，这个特征又叫做扩展方法，示例如下：



Formula接口在拥有calculate方法之外同时还定义了sqrt方法，实现了Formula接口的子类只需要实现一个calculate方法，默认方法sqrt将在子类上可以直接使用。

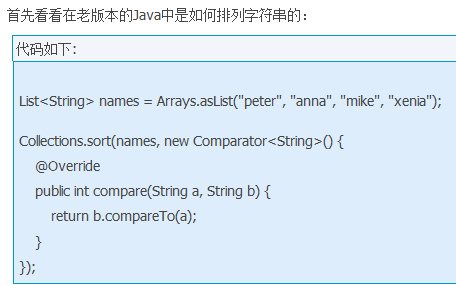


尽管默认方法非常强大，但是在使用默认方法时我们需要小心注意一个地方：在声明一个默认方法前，请仔细思考是不是真的有必要使用默认方法，因为默认方法会带给程序歧义，并且在复杂的继承体系中容易产生编译错误。



1. **Lambda 表达式**

Lambda是一个匿名函数，我们可以把Lambda表达式理解为是一段可以传递的代码（将代码像数据一样进行传递）。可以写出更简洁、更灵活的代码。作为一种更紧凑的代码风格，使Java的语音表达能力得到了提升。



只需要给静态方法 Collections.sort 传入一个List对象以及一个比较器来按指定顺序排列。通常做法都是创建一个匿名的比较器对象然后将其传递给sort方法。

在Java 8 中你就没必要使用这种传统的匿名对象的方式了，Java 8提供了更简洁的语法，lambda表达式：

Collections.sort(names, (String a, String b) -> {

return b.compareTo(a);

});

看到了吧，代码变得更段且更具有可读性，但是实际上还可以写得更短：

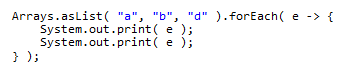
Collections.sort(names, (String a, String b) -> b.compareTo(a));

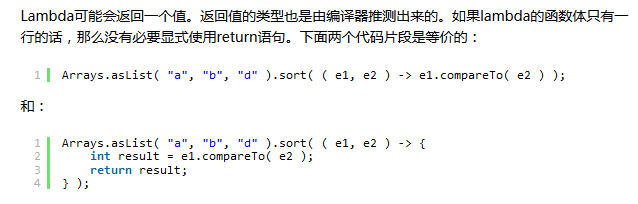
对于函数体只有一行代码的，你可以去掉大括号{}以及return关键字，但是你还可以写得更短点：

Collections.sort(names, (a, b) -> b.compareTo(a));

Java编译器可以自动推导出参数类型，所以你可以不用再写一次类型。接下来我们看看lambda表达式还能作出什么更方便的东西来.

在某些情况下lambda的函数体会更加复杂，这时可以把函数体放到在一对花括号中，就像在Java中定义普通函数一样。例如：





**注**：通常都会把lambda表达式内部变量的名字起得短一些。这样能使代码更简短，放在同一行。所以，在上述代码中，变量名选用a、b或者x、y会比even、odd要好。

**基本语法**

Lambda表达式的基础语法：Java8中引入了一个新的操作符“->”该操作符称为箭头操作符或Lambda操作符。箭头操作符将Lambda表达式分成两部分。

右侧：Lambda表达式中所需要执行的功能

左侧：表达式的参数列表

**语法格式**

1)、无参数，无返回值

()->System.out.println(55);

2)、有一个参数，无返回值;

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

如果只有一个参数的话，小括号可以省略

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

3)、如果lambda体中只有一条语句，大括号和return都可以省略

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

4）、有多个参数，有返回值，并且lambda体中有多条语句

(x,y)->{

System.out.println(x);

System.out.println(y);

};

5）、Lambda表达式的参数列表的数据类型可以省略不写，因为JVM编译器通过上下文推断出数据类型，即“类型推断”

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

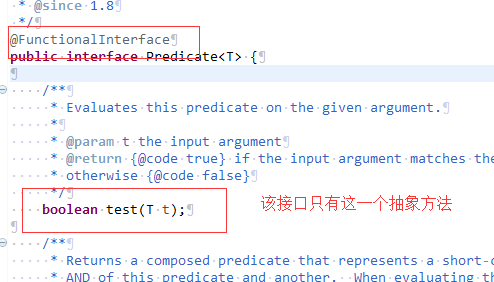
注：Lambda表达式需要函数式接口的支持

1. **函数式接口**

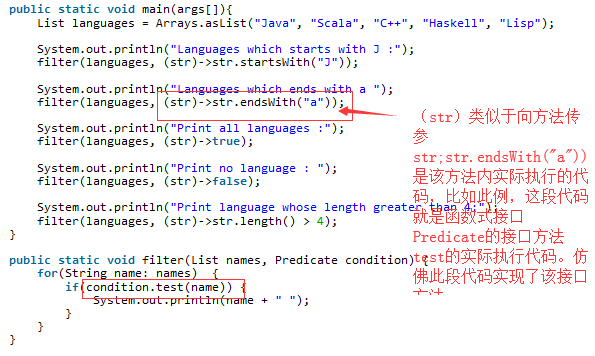
函数式接口就是只有一个抽象方法的普通接口。像这样的接口，可以被隐式转换为lambda表达式。java.lang.Runnable与java.util.concurrent.Callable是函数式接口最典型的两个。

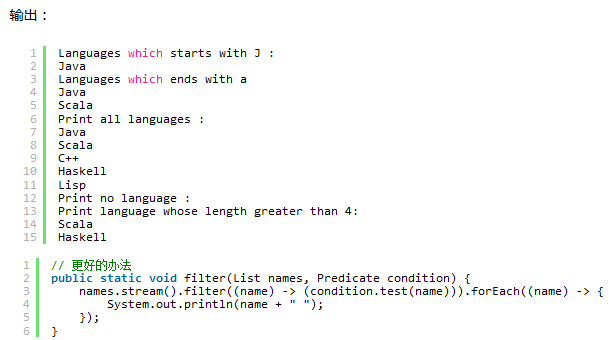
Lambda表达式是如何在java的类型系统中表示的呢？每一个lambda表达式都对应一个类型，通常是接口类型。而“函数式接口”是指仅仅只包含一个抽象方法的接口，每一个该类型的lambda表达式都会被匹配到这个抽象方法。因为默认方法不算抽象方法，所以你也可以给你的函数式接口添加默认方法。

我们可以将lambda表达式当作任意只包含一个抽象方法的接口类型，确保你的接口一定达到这个要求，你只需要给你的接口添加 @FunctionalInterface 注解，编译器如果发现你标注了这个注解的接口有多于一个抽象方法的时候会报错的。



除了在语言层面支持函数式编程风格，Java 8也添加了一个包，叫做 java.util.function。它包含了很多类，用来支持Java的函数式编程。其中一个便是Predicate，使用 java.util.function.Predicate 函数式接口以及lambda表达式，可以向API方法添加逻辑，用更少的代码支持更多的动态行为。

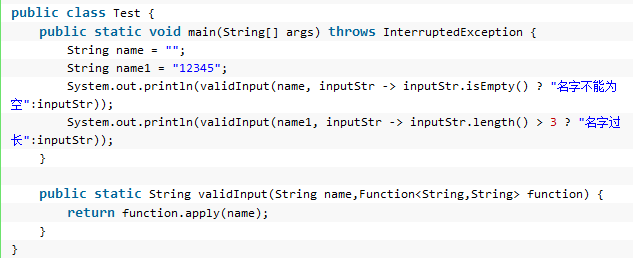




Java8内置的四大核心函数式接口，可以直接拿来用，如下：

**Function接口：**

接受一个参数，返回一个参数。



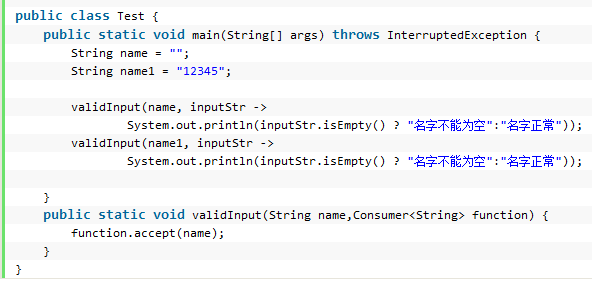
解释：

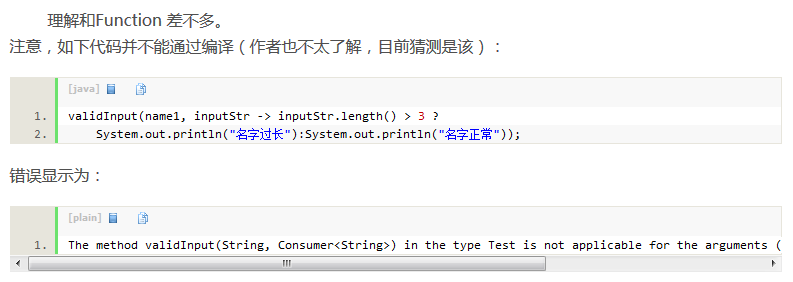
1. 定义 validInput 方法，传入 function 接口，然后在该方法中定义 function.apply(name)，也就是说，传入一个 name 参数，应用某些规则，返回一个结果，至于是什么规则，先不定义。

2.在main 方法中调用 validInput(name,inputStr ...)，这里我们定义规则，利用lambda 表达式， 规则是：传入一个 inputStr 字符串，如果为空，返回 xx；否则 返回 xx。

**Consumer接口：**

接受一个参数，不返回参数。

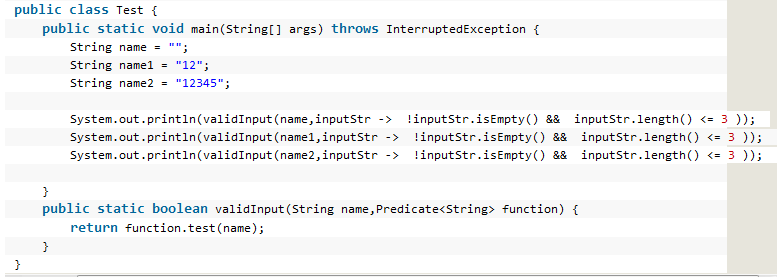




**Predicate接口：**

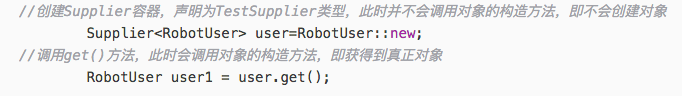
用于测试是否符合条件。

Predicate 方法 表示 判断 输入的对象是否 符合某个条件。

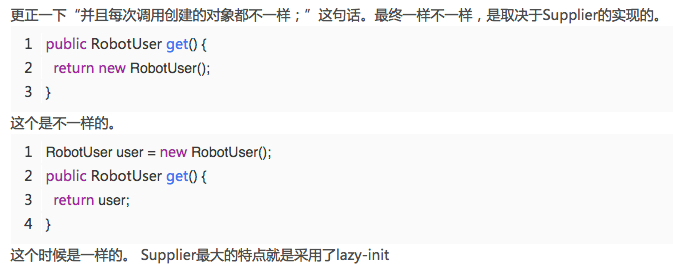


**Supplier接口：**

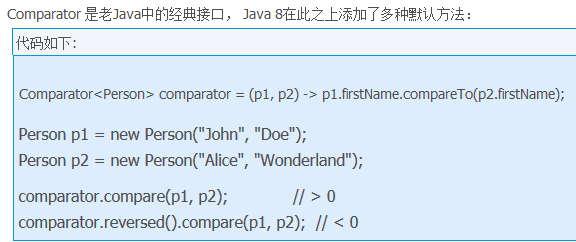
Supplier 接口返回一个任意范型的值，和Function接口不同的是该接口没有任何参数。



每次get都会调用构造方法，即获取的对象不同



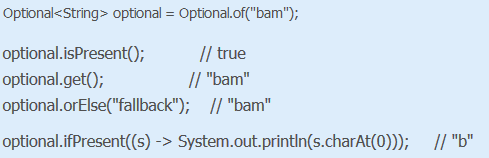
**Comparator接口：**



**Optional 接口：**

Optional 不是函数式接口，这是个用来防止NullPointerException异常的辅助类型，这是下一届中将要用到的重要概念，现在先简单的看看这个接口能干什么：

Optional 被定义为一个简单的容器，其值可能是null或者不是null。在Java 8之前一般某个函数应该返回非空对象但是偶尔却可能返回了null，而在Java 8中，不推荐你返回null而是返回Optional。

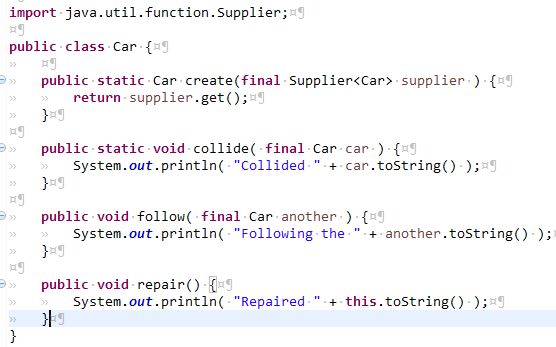


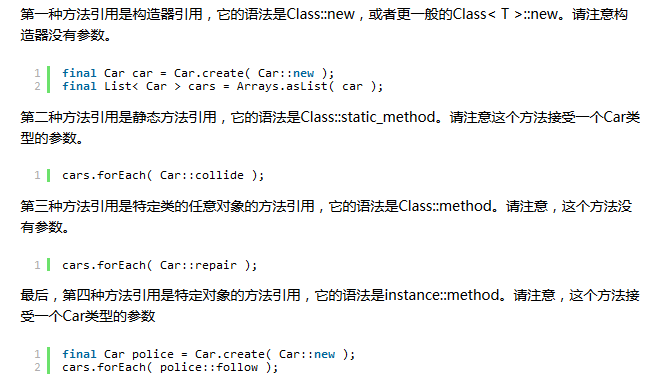
1. **方法与构造函数引用**

方法引用：若Lambda体中的内容有方法已经实现了，我们可以使用方法引用（可以理解为方法引用是Lambda表达式的另外一种表现形式）

方法引用提供了非常有用的语法，可以直接引用已有Java类或对象（实例）的方法或构造器。与lambda联合使用，方法引用可以使语言的构造更紧凑简洁，减少冗余代码。

条件：Lambda体中调用方法的参数列表与返回值类型，要与函数式接口中抽象方法的函数列表和返回值类型保持一致。





第一种对象::实例方法名

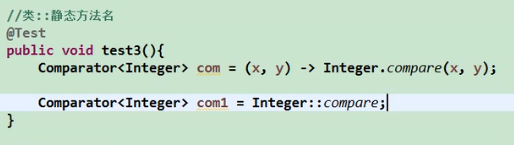




输出：

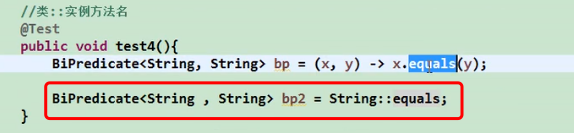


第二种：类::静态方法名



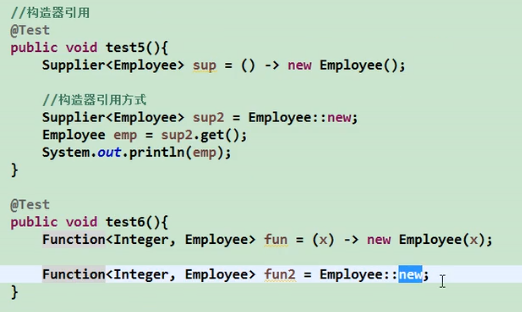
第三种：类::实例方法名

条件：若Lambda体参数列表第一个参数是实例方法的调用者，第二个参数是这个实例方法的参数时，就可以这样用

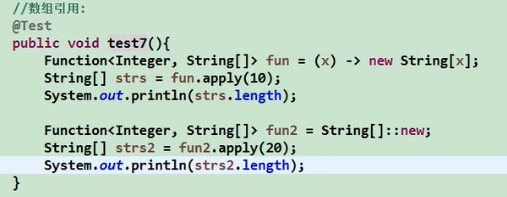


第四种：构造器引用

Employee有两个构造器，一个有参，一个无参，具体Employee::new调用的是有参还是无参，取决于Supplier和Function或者函数式接口里面的函数的参数情况。Supplier接口函数无参，则调用Employee无参构造器，Function接口函数有参，则调用Employee有参构造器。



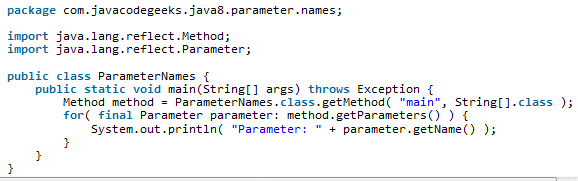
第五种：数组引用

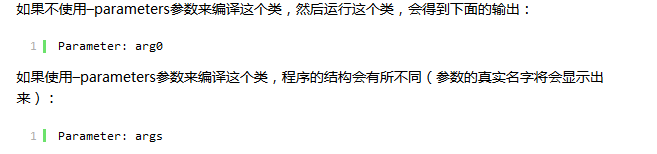


1. **Java编译器的新特性**

**参数名字:**

很长一段时间里，Java程序员一直在发明不同的方式使得[方法参数的名字能保留在Java字节码](http://www.javacodegeeks.com/2014/04/constructormethod-parameters-metadata-available-via-reflection-in-jdk-8.html" \t "_blank)中，并且能够在运行时获取它们（比如，[Paranamer类库](https://github.com/paul-hammant/paranamer" \t "_blank)）。最终，在Java 8中把这个强烈要求的功能添加到语言层面（通过反射API与Parameter.getName()方法）与字节码文件（通过新版的javac的–parameters选项）中。





对于有经验的**Maven**用户，通过maven-compiler-plugin的配置可以将-parameters参数添加到编译器中去。

<plugin>

    <groupId>org.apache.maven.plugins</groupId>

    <artifactId>maven-compiler-plugin</artifactId>

    <version>3.1</version>

    <configuration>

        <compilerArgument>-parameters</compilerArgument>

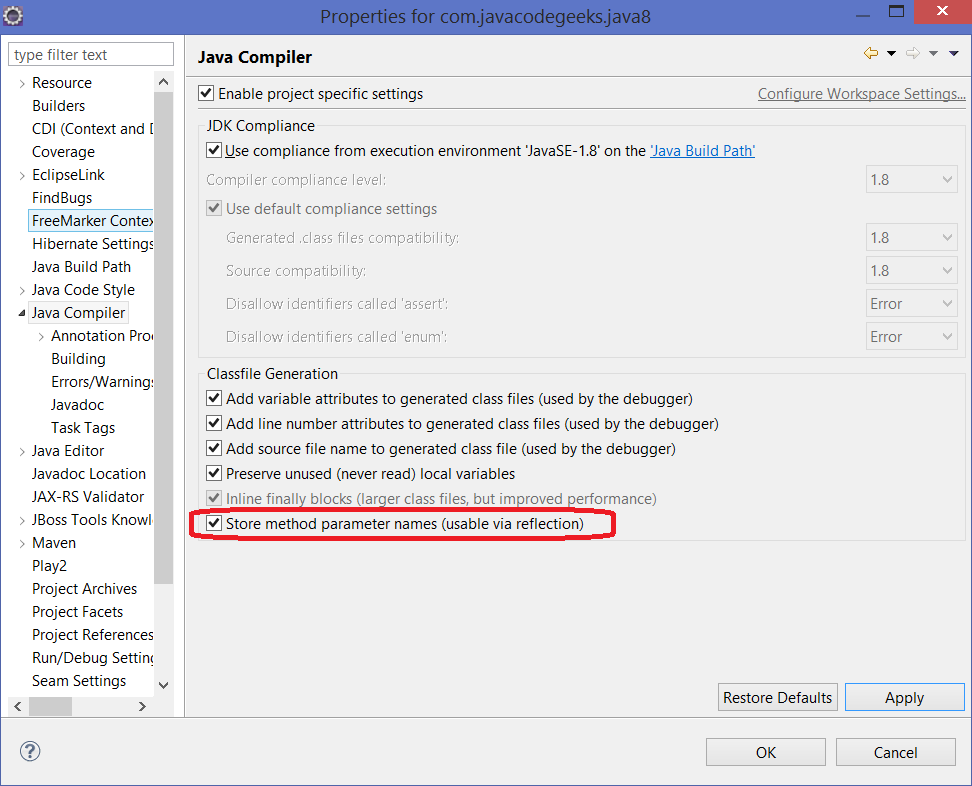
        <source>1.8</source>

        <target>1.8</target>

    </configuration>

</plugin>

针对Java 8最新发布的Eclipse Kepler SR2（请检查这里的下载说明）提供了非常实用的配置选项，可以通过下图的配置方式来控制编译器行为



此外，Parameter类有一个很方便的方法isNamePresent()来验证是否可以获取参数的名字。

1. **Stream API**

Stream是Java8中处理集合的关键抽象概念，它可以指定你希望对集合进行的操作，可以执行非常复杂的查找、过滤和映射数据等操作。使用StreamAPI对集合数据进行操作，就类似于使用SQL执行的数据库查询。也可以使用StreamAPI来并行执行操作，简而言之，Stream API提供了一种高效且易于使用的处理数据的方式。

流Stream到底是什么呢？

是数据渠道，用于操作数据源（集合、数组等）所生成的元素序列。

集合讲的是数据，流讲的是计算

注意：

Stream自己不会存储元素

Stream不会改变源对象，相反，他们会返回一个持有结果的新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

流的三个操作步骤

1.创建流

2.中间操作

3.终止操作(终端操作)

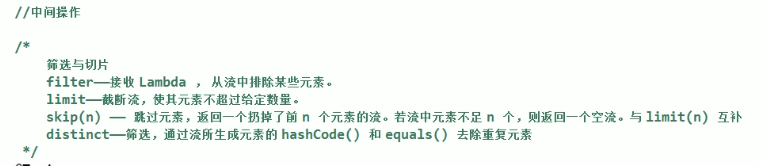
1. **创建流**

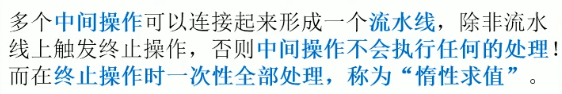


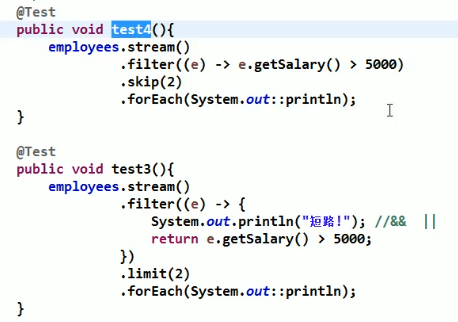
Java9还可以通过Optional的方式得到Stream流

|  |
| --- |
| List<String> list = **new** ArrayList<>(); list.add(**"Tom"**); list.add(**"Jerry"**); list.add(**"Tim"**);  Optional<List<String>> optional = Optional.*ofNullable*(list); Stream<List<String>> stream = optional.stream(); stream.flatMap(x -> x.stream()).forEach(System.***out***::println); |

1. **中间操作**







映射



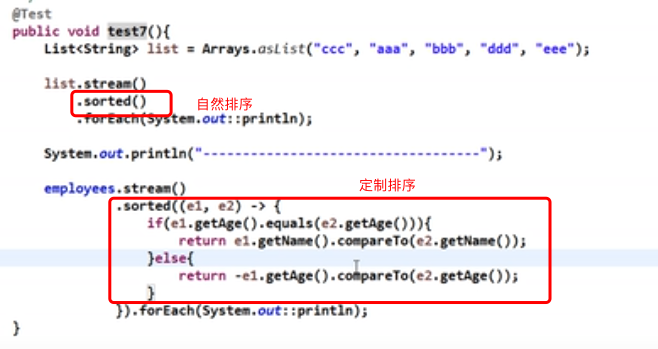
flatmap()方法是将流中的元素加入到流中。

map()方法是将流加入到流中

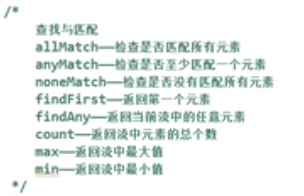
排序

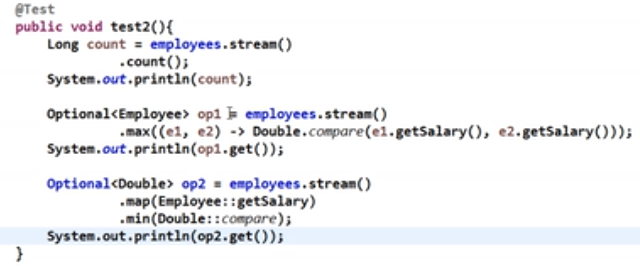
自然排序Comparable

定制排序Comparator



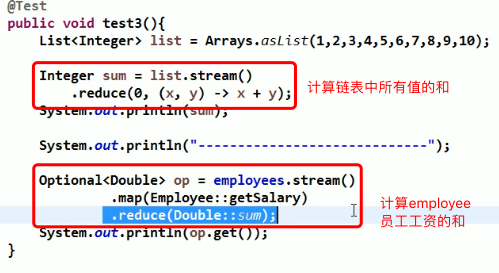
1. **终止操作**





归约

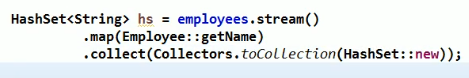




收集



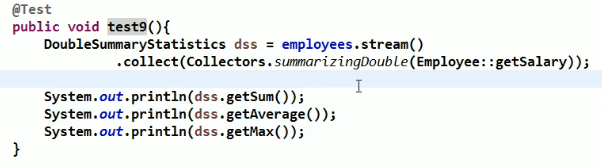
特殊集合



特殊收集



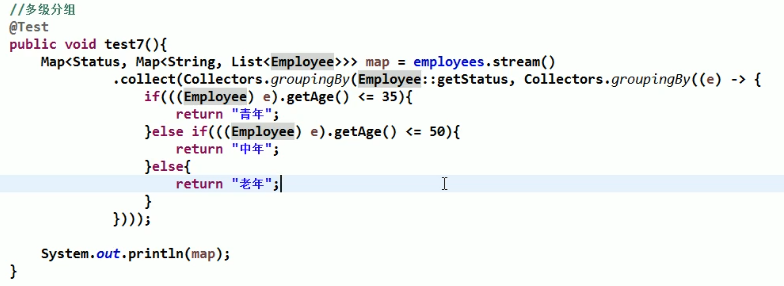
计算平均值、最大值、和



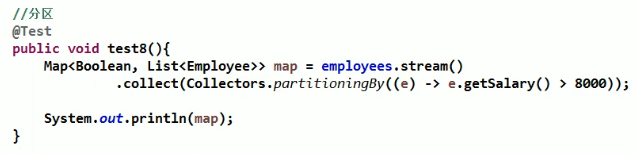
分组



多级分组

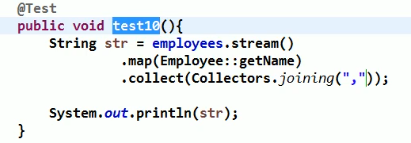


分区



true一个区，false一个区

连接字符串



中间添加,隔开

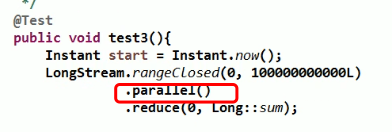
效果：



1. **并行流和顺序流**

并行流就是把一个内容分成多个数据块，并用不同的线程分别处理每个数据块的流。

Java8中将并行进行了优化，我们可以很容易的对数据进行并行操作。Stream API可以声明性地通过parallel()与sequential()在并行流与顺序流之间进行切换。



在大数据量进行计算处理的时候并行流肯定比顺序流快，但是数据量比较小，建议还是顺序流。

1. **Optional类**

Optional<T>类是一个容器类，代表一个值存在或不存在，原来用null表示一个值不存在，现在optional可以更好的表达这个概念，并且可以避免空指针异常。也有助于快速锁定空指针异常位置。

常用方法：

Optional.of(T t):创建一个Optional实例，如果t为null，直接报空指针异常，这样就能锁定空指针异常位置。

Optional.empty():创建一个空的Optional实例

get():获取实例

Optional.ofNullable(T t):若t不为null，创建Optional实例，否则创建空实例

isPresent()：判断是否包含值

orElse(T t):如果调用对象包含值，返回该值，否则返回t

orElseGet(Supplier s):如果调用对象包含值，返回该值，否则返回s获取的值。

map(Function f):如果有值对其处理，并返回处理后的Optional，否则返回Optional.empty()

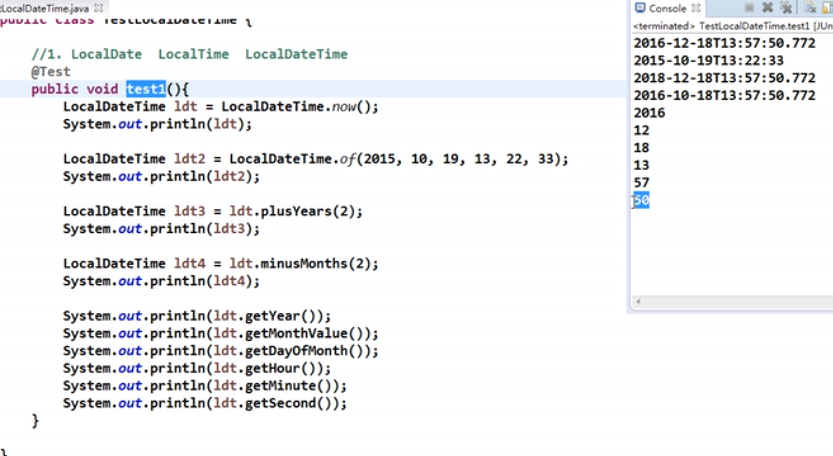
flatMap(Function mapper)：与map类似，要求返回值必须是Optional。

1. **新时间日期API**

由于旧日期API类的时区是可变的，因此存在多线程安全问题

LocalDate、LocalTime、LocalDateTime类的实例是不可变的对象，分别表示使用ISO-8601日历系统的日期、时间、日期和时间。它们提供了简单的日期或时间，并不包含当前的时间信息，也不包含与时区相关的信息。

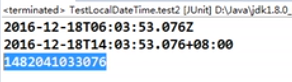
示例：



时间戳计算

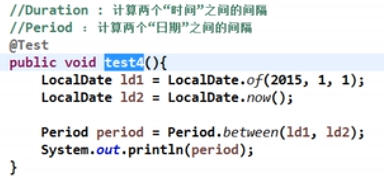


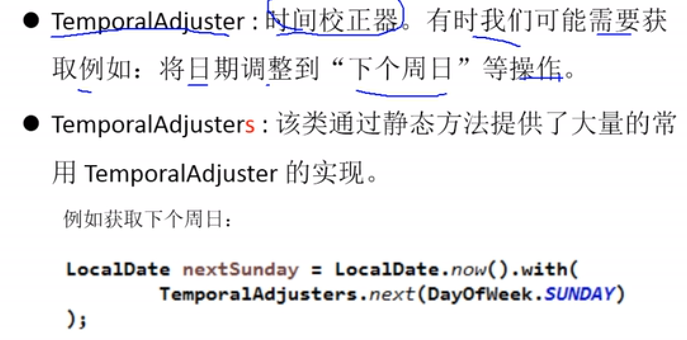
结果：



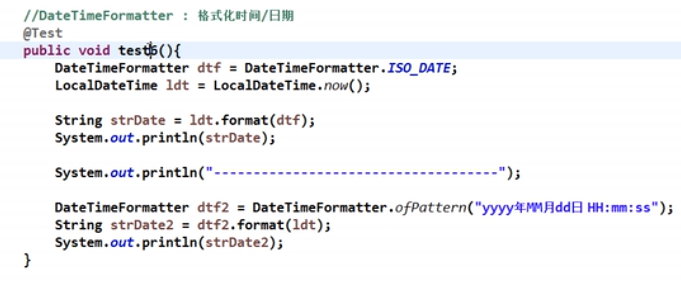
计算机当前时间







格式化日期



1. **重复注解与类型注解**