Java Lambda表达式入门

2014年04月27日 21:17:58 铁锚 阅读数：187024 标签： java java8 lambda stream Lambda表达式 更多
个人分类： Java基础

原文链接: **Start Using Java Lambda Expressions**

下载示例程序 **Examples.zip** 。
原文日期: 2014年4月16日

翻译日期: 2014年4月27日
翻译人员: **铁锚**

简介

(**译者认为**: **超过3行的逻辑就不适用Lambda表达式了**。虽然看着很先进，其实Lambda表达式的本质只是一个"**语法糖**"，由编译器在编译阶段将你转换包装为常规的代码,因此你可以使用更少的代码来实现同样的功能。本人建议不要乱用,因为这就和某些很高级的语法一样,会导致代码简洁,难懂,难以调试,维护人员想骂娘.)

Lambda表达式是Java SE 8中一个重要的新特性。lambda表达式允许你通过表达式来代替功能接口。 lambda表达式提供了一个正常的参数列表和一个使用这些参数的主体(body,可以是一个表达式或一个代码块)。

Lambda表达式还增强了集合库。 Java SE 8添加了2个对集合数据进行批量操作的包: java.util.function 包以及 java.ut (stream)就如同迭代器(iterator),但附加了许多额外的功能。 总的来说,lambda表达式和 stream 是自Java语言添加泛型(annotation)以来最大的变化。 在本文中,我们将从简单到复杂的示例中见认识lambda表达式和stream的强悍。

环境准备

如果还没有安装Java 8,那么你应该先安装才能使用lambda和stream(译者建议在**虚拟机**中安装,测试使用)。 像NetBean类的工具和IDE就支持Java 8特性,包括lambda表达式,可重复的注解,紧凑的概要文件和其他特性。

下面是Java SE 8和NetBeans IDE 8的下载链接:

Java Platform (JDK 8): 从Oracle下载Java 8,也可以和NetBeans IDE一起下载
NetBeans IDE 8: 从NetBeans官网下载NetBeans IDE

Lambda表达式的语法

基本语法:

(parameters) -> expression

或

(parameters) ->{ statements; }

下面是Java lambda表达式的简单例子:

```
1  // 1. 不需要参数,返回值为 5
2  () -> 5
3
4  // 2. 接收一个参数(数字类型),返回其2倍的值
5  x -> 2 * x
6
7  // 3. 接受2个参数(数字),并返回他们的差值
8  (x, y) -> x – y
9
10 // 4. 接收2个int型整数,返回他们的和
11 (int x, int y) -> x + y
12
13 // 5. 接受一个 string 对象,并在控制台打印,不返回任何值(看起来像是返回void)
14 (String s) -> System.out.print(s)
```

基本的Lambda例子

现在,我们已经知道什么是lambda表达式,让我们先从一些基本的例子开始。 在本节中,我们将看到lambda表达式如何

式。 假设有一个玩家List ,程序员可以使用 for 语句 ("for 循环")来遍历,在Java SE 8中可以转换为另一种形式:

```
String[] atp = {"Rafael Nadal", "Novak Djokovic",
       "Stanislas Wawrinka",
       "David Ferrer","Roger Federer",
       "Andy Murray","Tomas Berdych",
       "Juan Martin Del Potro"};
List<String> players =  Arrays.asList(atp);

// 以前的循环方式
for (String player : players) {
     System.out.print(player + "; ");
}

// 使用 lambda 表达式以及函数操作(functional operation)
players.forEach((player) -> System.out.print(player + "; "));

// 在 Java 8 中使用双冒号操作符(double colon operator)
players.forEach(System.out::println);
```

正如您看到的,lambda表达式可以将我们的代码缩减到一行。 另一个例子是在图形用户界面程序中,匿名类可以使用lar替。 同样,在实现Runnable接口时也可以这样使用:

```
// 使用匿名内部类
btn.setOnAction(new EventHandler<ActionEvent>() {
          @Override
          public void handle(ActionEvent event) {
              System.out.println("Hello World!");
          }
    });

// 或者使用 lambda expression
btn.setOnAction(event -> System.out.println("Hello World!"));
```

下面是使用lambdas 来实现 Runnable接口 的示例:

```
// 1.1使用匿名内部类
new Thread(new Runnable() {
    @Override
    public void run() {
        System.out.println("Hello world !");
    }
}).start();

// 1.2使用 lambda expression
new Thread(() -> System.out.println("Hello world !")).start();

// 2.1使用匿名内部类
Runnable race1 = new Runnable() {
    @Override
    public void run() {
        System.out.println("Hello world !");
```

```
    }18 };

// 2.2使用 lambda expression
Runnable race2 = () -> System.out.println("Hello world !");

// 直接调用 run 方法(没开新线程哦!)
race1.run();
race2.run();
```

Runnable 的 lambda表达式,使用块格式,将五行代码转换成单行语句。 接下来,在下一节中我们将使用lambdas对集合

使用Lambdas排序集合

在Java中,Comparator 类被用来排序集合。 在下面的例子中,我们将根据球员的 name, surname, name 长度 以及最后的示例一样,先使用匿名内部类来排序,然后再使用lambda表达式精简我们的代码。

在第一个例子中,我们将根据name来排序list。 使用旧的方式,代码如下所示:

```
String[] players = {"Rafael Nadal", "Novak Djokovic",
    "Stanislas Wawrinka", "David Ferrer",
    "Roger Federer", "Andy Murray",
    "Tomas Berdych", "Juan Martin Del Potro",
    "Richard Gasquet", "John Isner"};

// 1.1 使用匿名内部类根据 name 排序 players
Arrays.sort(players, new Comparator<String>() {
    @Override
    public int compare(String s1, String s2) {
        return (s1.compareTo(s2));
    }
});
```

使用lambdas,可以通过下面的代码实现同样的功能:

```
// 1.2 使用 lambda expression 排序 players
Comparator<String> sortByName = (String s1, String s2) -> (s1.compareTo(s2));
Arrays.sort(players, sortByName);

// 1.3 也可以采用如下形式:
Arrays.sort(players, (String s1, String s2) -> (s1.compareTo(s2)));
```

其他的排序如下所示。 和上面的示例一样,代码分别通过匿名内部类和一些lambda表达式来实现Comparator :

```
// 1.1 使用匿名内部类根据 surname 排序 players
Arrays.sort(players, new Comparator<String>() {
    @Override
    public int compare(String s1, String s2) {
        return (s1.substring(s1.indexOf(" ")).compareTo(s2.substring(s2.indexOf(" "))));
    }
});

// 1.2 使用 lambda expression 排序,根据 surname
```

```
Comparator<String> sortBySurname = (String s1, String s2) ->
                                                                11
    ( s1.substring(s1.indexOf(" ")).compareTo( s2.substring(s2.indexOf(" ")) ) );12
Arrays.sort(players, sortBySurname);13
// 1.3 或者这样,怀疑原作者是不是想错了,括号好多...
Arrays.sort(players, (String s1, String s2) ->
      ( s1.substring(s1.indexOf(" ")).compareTo( s2.substring(s2.indexOf(" ")) ) )
    );

// 2.1 使用匿名内部类根据 name lenght 排序 players
Arrays.sort(players, new Comparator<String>() {
    @Override
    public int compare(String s1, String s2) {
        return (s1.length() - s2.length());
    }
});

// 2.2 使用 lambda expression 排序,根据 name lenght
Comparator<String> sortByNameLenght = (String s1, String s2) -> (s1.length() - s2.length());
Arrays.sort(players, sortByNameLenght);

// 2.3 or this
Arrays.sort(players, (String s1, String s2) -> (s1.length() - s2.length()));

// 3.1 使用匿名内部类排序 players, 根据最后一个字母
Arrays.sort(players, new Comparator<String>() {
    @Override
    public int compare(String s1, String s2) {
        return (s1.charAt(s1.length() - 1) - s2.charAt(s2.length() - 1));
    }
});

// 3.2 使用 lambda expression 排序,根据最后一个字母
Comparator<String> sortByLastLetter =
    (String s1, String s2) ->
        (s1.charAt(s1.length() - 1) - s2.charAt(s2.length() - 1));
Arrays.sort(players, sortByLastLetter);

// 3.3 or this
Arrays.sort(players, (String s1, String s2) -> (s1.charAt(s1.length() - 1) - s2.charAt(s2.le
```

就是这样,简洁又直观。 在下一节中我们将探索更多lambdas的能力,并将其与 stream 结合起来使用。

使用Lambdas和Streams

Stream是对集合的包装,通常和lambda一起使用。 使用lambdas可以支持许多操作,如 map, filter, limit, sorted, count, r
collect 等等。 同样,Stream使用**懒运算**,他们并不会真正地读取所有数据,遇到像getFirst() 这样的方法就会结束链式语法
中,我们将探索lambdas和streams 能做什么。 我们创建了一个Person类并使用这个类来添加一些数据到list中,将用于这
Person 只是一个简单的POJO类:

```
public class Person {

private String firstName, lastName, job, gender;
private int salary, age;

public Person(String firstName, String lastName, String job,
                String gender, int age, int salary)       {
          this.firstName = firstName;
          this.lastName = lastName;
          this.gender = gender;
```

```
        this.age = age;
        this.job = job;
        this.salary = salary;
}
// Getter and Setter
// . . . . .
}
```

接下来,我们将创建两个list,都用来存放Person对象:

```
List<Person> javaProgrammers = new ArrayList<Person>() {
  {
    add(new Person("Elsdon", "Jaycob", "Java programmer", "male", 43, 2000));
    add(new Person("Tamsen", "Brittany", "Java programmer", "female", 23, 1500));
    add(new Person("Floyd", "Donny", "Java programmer", "male", 33, 1800));
    add(new Person("Sindy", "Jonie", "Java programmer", "female", 32, 1600));
    add(new Person("Vere", "Hervey", "Java programmer", "male", 22, 1200));
    add(new Person("Maude", "Jaimie", "Java programmer", "female", 27, 1900));
    add(new Person("Shawn", "Randall", "Java programmer", "male", 30, 2300));
    add(new Person("Jayden", "Corrina", "Java programmer", "female", 35, 1700));
    add(new Person("Palmer", "Dene", "Java programmer", "male", 33, 2000));
    add(new Person("Addison", "Pam", "Java programmer", "female", 34, 1300));
  }
};

List<Person> phpProgrammers = new ArrayList<Person>() {
  {
    add(new Person("Jarrod", "Pace", "PHP programmer", "male", 34, 1550));
    add(new Person("Clarette", "Cicely", "PHP programmer", "female", 23, 1200));
    add(new Person("Victor", "Channing", "PHP programmer", "male", 32, 1600));
    add(new Person("Tori", "Sheryl", "PHP programmer", "female", 21, 1000));
    add(new Person("Osborne", "Shad", "PHP programmer", "male", 32, 1100));
    add(new Person("Rosalind", "Layla", "PHP programmer", "female", 25, 1300));
    add(new Person("Fraser", "Hewie", "PHP programmer", "male", 36, 1100));
    add(new Person("Quinn", "Tamara", "PHP programmer", "female", 21, 1000));
    add(new Person("Alvin", "Lance", "PHP programmer", "male", 38, 1600));
    add(new Person("Evonne", "Shari", "PHP programmer", "female", 40, 1800));
  }
};
```

现在我们使用forEach方法来迭代输出上述列表:

```
System.out.println("所有程序员的姓名:");
javaProgrammers.forEach((p) -> System.out.printf("%s %s; ", p.getFirstName(), p.getLastName(
phpProgrammers.forEach((p) -> System.out.printf("%s %s; ", p.getFirstName(), p.getLastName()
```

我们同样使用forEach方法,增加程序员的工资5%:

```
System.out.println("给程序员加薪 5% :");
Consumer<Person> giveRaise = e -> e.setSalary(e.getSalary() / 100 * 5 + e.getSalary());

javaProgrammers.forEach(giveRaise);
```

```
5 phpProgrammers.forEach(giveRaise);
```

另一个有用的方法是过滤器filter() ,让我们显示月薪超过1400美元的PHP程序员:

```
1 System.out.println("下面是月薪超过 $1,400 的PHP程序员:")
2 phpProgrammers.stream()
3           .filter((p) -> (p.getSalary() > 1400))
4           .forEach((p) -> System.out.printf("%s %s; ", p.getFirstName(), p.getLastName()));
```

我们也可以定义过滤器,然后重用它们来执行其他操作:

```
1  // 定义 filters
2  Predicate<Person> ageFilter = (p) -> (p.getAge() > 25);
3  Predicate<Person> salaryFilter = (p) -> (p.getSalary() > 1400);
4  Predicate<Person> genderFilter = (p) -> ("female".equals(p.getGender()));
5
6  System.out.println("下面是年龄大于 24岁且月薪在$1,400以上的女PHP程序员:");
7  phpProgrammers.stream()
8            .filter(ageFilter)
9            .filter(salaryFilter)
10           .filter(genderFilter)
11           .forEach((p) -> System.out.printf("%s %s; ", p.getFirstName(), p.getLastName()));
12
13 // 重用filters
14 System.out.println("年龄大于 24岁的女性 Java programmers:");
15 javaProgrammers.stream()
16           .filter(ageFilter)
17           .filter(genderFilter)
18           .forEach((p) -> System.out.printf("%s %s; ", p.getFirstName(), p.getLastName()));
```

使用limit方法,可以限制结果集的个数:

```
1  System.out.println("最前面的3个 Java programmers:");
2  javaProgrammers.stream()
3            .limit(3)
4            .forEach((p) -> System.out.printf("%s %s; ", p.getFirstName(), p.getLastName()));
5
6
7  System.out.println("最前面的3个女性 Java programmers:");
8  javaProgrammers.stream()
9            .filter(genderFilter)
10           .limit(3)
11           .forEach((p) -> System.out.printf("%s %s; ", p.getFirstName(), p.getLastName()));
```

排序呢? 我们在stream中能处理吗? 答案是肯定的。 在下面的例子中,我们将根据名字和薪水排序Java程序员,放到一个表:

// 静态引入

import static java.util.stream.Collectors.toList;

```
System.out.println("根据 name 排序,并显示前5个 Java programmers:");
List<Person> sortedJavaProgrammers = javaProgrammers
          .stream()
          .sorted((p, p2) -> (p.getFirstName().compareTo(p2.getFirstName())))
          .limit(5)
          .collect(toList());

sortedJavaProgrammers.forEach((p) -> System.out.printf("%s %s; %n", p.getFirstName(), p.getL

System.out.println("根据 salary 排序 Java programmers:");
sortedJavaProgrammers = javaProgrammers
          .stream()
          .sorted( (p, p2) -> (p.getSalary() - p2.getSalary()) )
          .collect( toList() );

sortedJavaProgrammers.forEach((p) -> System.out.printf("%s %s; %n", p.getFirstName(), p.getL
```

如果我们只对最低和最高的薪水感兴趣,比排序后选择第一个/最后一个 更快的是min和max方法:

```
System.out.println("工资最低的 Java programmer:");
Person pers = javaProgrammers
          .stream()
          .min((p1, p2) -> (p1.getSalary() - p2.getSalary()))
          .get()

System.out.printf("Name: %s %s; Salary: $%,d.", pers.getFirstName(), pers.getLastName(), pers.ge

System.out.println("工资最高的 Java programmer:");
Person person = javaProgrammers
          .stream()
          .max((p, p2) -> (p.getSalary() - p2.getSalary()))
          .get()

System.out.printf("Name: %s %s; Salary: $%,d.", person.getFirstName(), person.getLastName(), per
))
```

上面的例子中我们已经看到 collect 方法是如何工作的。 结合 map 方法,我们可以使用 collect 方法来将我们的结果集放
Set 或一个TreeSet中:

```
System.out.println("将 PHP programmers 的 first name 拼接成字符串:");
String phpDevelopers = phpProgrammers
          .stream()
          .map(Person::getFirstName)
          .collect(joining(" ; ")); // 在进一步的操作中可以作为标记(token)

System.out.println("将 Java programmers 的 first name 存放到 Set:");
Set<String> javaDevFirstName = javaProgrammers
          .stream()
          .map(Person::getFirstName)
          .collect(toSet());

System.out.println("将 Java programmers 的 first name 存放到 TreeSet:");
```

```
14 TreeSet<String> javaDevLastName = javaProgrammers15           .stream()
16           .map(Person::getLastName)
17           .collect(toCollection(TreeSet::new));
```

Streams 还可以是并行的(parallel)。 示例如下:

```
1 System.out.println("计算付给 Java programmers 的所有money:");
2 int totalSalary = javaProgrammers
3           .parallelStream()
4           .mapToInt(p -> p.getSalary())
5           .sum();
```

我们可以使用summaryStatistics方法获得stream 中元素的各种汇总数据。 接下来,我们可以访问这些方法,比如getMax getAverage:

```
1 //计算 count, min, max, sum, and average for numbers
2 List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
3 IntSummaryStatistics stats = numbers
4           .stream()
5           .mapToInt((x) -> x)
6           .summaryStatistics();
7
8 System.out.println("List中最大的数字 : " + stats.getMax());
9 System.out.println("List中最小的数字 : " + stats.getMin());
10 System.out.println("所有数字的总和   : " + stats.getSum());
11 System.out.println("所有数字的平均值 : " + stats.getAverage());
```

OK,就这样,希望你喜欢它!

总结

在本文中,我们学会了使用lambda表达式的不同方式,从基本的示例,到使用lambdas和streams的复杂示例。 此外,我们 lambda表达式与Comparator 类来对Java集合进行排序。

CSS高级技巧:自动省略左侧文本
WebRTC基础实践 - 10. 总结

博主专栏

GC性能优化

文章数：18 篇　访问量：26万+

有问题？加入开源技术交流群

热门文章

Windows下安装并设置Redis
阅读数 300863
IntelliJ IDEA 设置代码提示或自动补全的快捷键
阅读数 267600
创建GitHub技术博客全攻略
阅读数 237855
Java Lambda表达式入门
阅读数 186991
3. Git与TortoiseGit基本操作
阅读数 173426

个人分类

HTML5_前端 53篇
面试笔试 6篇
Java基础 49篇
JS笔记 31篇
想法 8篇
展开

归档

2019年3月 2篇
2019年2月 1篇
2018年12月 4篇
2018年11月 5篇
2018年10月 2篇
展开

ambda表达式究竟需要怎么用 阅读数 2384

已经有一段时间了，这次发布的改动比较大，很多人将这次改动与Java5的升级… 博文 来自： fac…

ambda使用总结－结合实例介绍 阅读数 8107

va8新特性介绍2.项目中应用1.Java8新特性介绍原文链接 https://my.oschina.net/c… 博文 来自： 雨…

mbda表达式的应用 阅读数 859

表达式λ表达式本质上是一个匿名方法。让我们来看下面这个例子： publicintadd… 博文 来自： 木…

va8新特性Lamb表达式简单了解 阅读数 1230

新的项目，jdk使用的是1.8+，看了一下源码，源码中有些地方用了java8的一些新… 博文 来自： zo…

ambuda表达式 阅读数 1348

算符：所有的lambda表达式都是用新的lambda运算符"=>",可以叫他，“转到”或者“… 博文 来自： liuji…

amb表达式 阅读数 112

a8提供了默认接口方法，Lambda表达式，方法引用和重复注解等新的特性及API2… 博文 来自： Fait…

ambda详解 阅读数 8681

参数在Lambda表达式中，输入参数是Lambda运算符的左边部分。它包含参数的… 博文 来自： wxy…

ambda 表达式的优缺点总结 阅读数 1万+

到lambda表达式后，发现它基本上可以消灭所有for循环，换句话说，所有的for循… 博文 来自： 天…

Lambda表达式10个示例 阅读数 2278

mbda表达式实现Runnable我开始使用Java8时，首先做的就是使用lambda表达式… 博文 来自： 零…

mbda表达式 阅读数 753

p://www.cnblogs.com/WJ5888/p/4618465.html1.1 引言课本上说编程有两种模式… 博文 来自： 正…

— Lambada 表达式 阅读数 194

数又称匿名函数，匿名函数就是没有名字的函数。引例、仅用于理解Lambda>>>… 博文 来自： star…

a表达式-使用说明 阅读数 1120

jdk8已经发布4年，其中有一个特性：Lambda，它是一个令开发者便捷开发… 博文 来自： Luk…

Lambda 使用实例 阅读数 1044

对Java8发布的Lambda表达式进行一定了解之后，发现Lambda最核心的就是结… 博文 来自： 我…

ambd表达式学习（一） 阅读数 534

要Lambda表达式?学习之前先问为什么，这是惯性思维使然，下面从优缺点来分… 博文 来自： wzg…

nbad表达式 阅读数 1259

表达式 λ表达式本质上是一个匿名方法。让我们来看下面这个例子： publicintad… 博文 来自： Hap…

lambda表达式 阅读数 231

布四年之久，但是一直没研究java8的相关新特性，前几天有点闲，就学习了java… 博文 来自： xiao…

ambda表达式10个示例 阅读数 1203

了lambda表达式，对于lambda表达式和API。越来越多的了解它们，越能够写出… 博文 来自： lon…

ambda表达式 实现原理分析 阅读数 3741

基于JDK9一、目标本文主要解决两个问题：1、函数式接口到底是什么？2、Lamb… 博文 来自： 衣…

ambda表达式的用法 阅读数 424
去: Lambda 表达式的写法: 最简单的用法:简化匿名内部类. 1. 如果要重写的方... 博文 来自： Che...

ambda表达式的几个示例 阅读数 1464
mbda表达式实现Runnable我开始使用Java8时，首先做的就是使用lambda表达式... 博文 来自： 黎...

Lambda表达式入门 阅读数 3081
nda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... 博文 来自： 晏...

使用lambda表达式进行集合的遍历 阅读数 51
会用到各种集合，数字的，字符串的还有对象的。它们无处不在，哪怕操作集合的... 博文 来自： 浅...

nbda表达式 阅读数 59
说Java8将要支持Lambda表达式（或称闭包），我便开始狂热的想要将这些体面... 博文 来自： 珍惜

中的lambda表达式 阅读数 31
始设计的是完全面向对象的,所以一等公民都是对象.函数只是作为对象上的附属.而... 博文 来自： 大...

读Java8-lambda表达式之方法引用 阅读数 1684
子importjava.util.ArrayList;importjava.util.Arrays;importstaticjava.util.Comparator.c... 博文 来自： won...

表达式的10个示例 阅读数 3141
Java8lambda表达式10个示例Java8刚于几周前发布，日期是2014年3月18日，这... 博文 来自： zhe...

ambda表达式及方法引用 阅读数 1336
ambda表达式是JavaSE8中一个重要的新特性。允许你通过表达式来代替功能接... 博文 来自： 一...

系列之Lambda表达式 阅读数 5704
ambda表达式也有一段时间了，有时候用的云里雾里的，是该深入学习Java8新... 博文 来自： 行

java8lambda表达式原理java8已经推出有一段时间了，相信有不少公司已经把jdk升级到8了... 博文 来自： vall...

JAVA8自定义Lambda表达式的常见使用方法 阅读数 1784
根据使用场景不同，Lambda表达式的常见使用方法通常有三种写法。总结：当某个类在需... 博文 来自： 王...

Java8-lambda表达式 阅读数 1203
在EclipseIDE中试用Lambda表达式作者：DeepakVohra学习如何充分利用lambda和虚拟扩... 博文 来自： sha...

Java8 lambda表达式常用方法 阅读数 1027
闲话不多说，直接上代码。先定义一个用户类。classUser{Integerid;Stringname;publicInteg... 博文

Java8中的Lambda表达式 阅读数 1732
一.简述Lambda表达式本是属于JDK1.7的Lambda项目的内容.在JDK1.7开发琪琪,Sun公司... 博文 来自： 懒...

Java8 新特性 lambda表达式详解 阅读数 6334
java8新特性lambda表达式详解 博文 来自： xiao...

一、java8的Lambda表达式 阅读数 1842
什么是Lambda表达式Lambda表达式是一段可以传递的代码。λ表达式本质上是一个匿名方... 博文 来自： 等...

Java8 Lambda 表达式与 Checked Exception 阅读数 2450
当我们在使用Java8的Lambda表达式时，表达式内容需要抛出异常，也许还会想当然的让… 博文 来自： 一…

关于java中Lambda表达式的使用（粗解） 阅读数 1968
Lambda表达式的使用可以很大程度上减少代码的数量，但是阅读起来并不是怎么方便，如… 博文 来自： 朱…

Java中Lambda表达式的使用 阅读数 396
博客转自：http://www.cnblogs.com/franson-2016/p/5593080.html简介(译者注:虽然看着很… 博文 来自： bru…

JAVA8中Lambda表达式入门 阅读数 94
http://blog.csdn.net/joker_honey/article/details/54970179原文Lambda表达式是JavaSE8中… 博文 来自： qq_…

JAVA8之lambda表达式详解，及stream中的lambda使用 阅读数 3万+
1.什么是lambda表达式？2.lambda表达式用来干什么的？3.lambda表达式的优缺点？4.lam… 博文 来自： 凤歌

java8学习笔记1（Lambda表达式） 阅读数 71
Lambda表达式Lambda表达式，也可称为闭包，它是推动Java8发布的最重要新特性。Lam… 博文 来自： IT…

函数式接口和lambda表达式 阅读数 3829
函数式接口和lambda表达式函数式接口(FunctionalInterface)：任何接口，如果只包含唯一… 博文 来自： alw…

java8之接口的默认方法和lambda表达式小结(一) 阅读数 499
java8之小小结(一) 博文 来自： Ste…

Java中的lamda表达式---更加高效简洁的表达方式 阅读数 2474
前言刚好今天看到了lambda表达式，学习了一下基本用法，由于上午就是翻开几篇博客看… 博文 来自： 潘…

Java lambda表达式 阅读数 86
今天正儿八经地学习一下java8,这里记录一下.(我看的书是《java8实战》，这里举的例子大… 博文 来自： 阿…

Java8 Lambda表达式 阅读数 71
什么是lambda表达式：lambda表达式是一个匿名函数，即没有函数名的函数。Java8使用l… 博文 来自： Em…

Java8-Lambda表达式 阅读数 207
Lambda表达式，它可以很简洁地表示一个行为或传递代码，现在你可以把Lambda表达式… 博文 来自： Gh…

java Lambda表达式的使用 阅读数 417
如题，因为博主也是最近才接触到Lambda表达式的（PS在这里汗颜一会）。我并不会讲解… 博文 来自： plani

数据标准化/归一化normalization 阅读数 19万+
http://blog.csdn.net/pipisorry/article/details/52247379 这里主要讲连续型特征归一化的常用… 博文 来自： 皮…

expat介绍文档翻译 阅读数 4万+
原文地址：http://www.xml.com/pub/a/1999/09/expat/index.html 因为需要用，所以才翻译了… 博文 来自： ymj…

R语言｜文本挖掘之中文分词包——Rwordseg包(原理、功能、详解) 阅读数 3万+
笔者寄语：与前面的RsowballC分词不同的地方在于这是一个中文的分词包，简单易懂，分… 博文 来自： 素…

redis两种持久化策略 阅读数 2万+
reids是一个key-value存储系统，为了保证效率，缓存在内存中，但是redis会周期性的把更… 博文 来自： 那…

DirectX修复工具增强版 阅读数 195万+

最后更新：2018-12-20 DirectX修复工具最新版：DirectX Repair V3.8 增强版 NEW! 版本… 博文 来自： VBc…

搭建图片服务器《二》-linux安装nginx 阅读数 4万+

nginx是个好东西，Nginx (engine x) 是一个高性能的HTTP和反向代理服务器，也是一个IM… 博文 来自： ma…

非局部均值去噪（NL-means） 阅读数 1万+

非局部均值（NL-means）是近年来提出的一项新型的去噪技术。该方法充分利用了图像中… 博文 来自： xiao…

多重背包O(N*V)算法详解（使用单调队列） 阅读数 2万+

多重背包问题：有N种物品和容量为V的背包，若第i种物品，容量为v[i]，价值为w[i]，共有n… 博文 来自： flyin…

机器学习-数据归一化 阅读数 5875

定义 数据标准化（归一化）处理是数据挖掘的一项基础工作，不同评价指标往往具有不同… 博文 来自： yeh…

高斯混合模型（GMM）及其EM算法的理解 阅读数 10万+

一个例子高斯混合模型（Gaussian Mixed Model）指的是多个高斯分布函数的线性组合，… 博文 来自： 小…

jquery/js实现一个网页同时调用多个倒计时(最新的) 阅读数 45万+

jquery/js实现一个网页同时调用多个倒计时(最新的) 最近需要网页添加多个倒计时. 查阅网… 博文 来自： We…

关于SpringBoot bean无法注入的问题（与文件包位置有关） 阅读数 18万+

问题场景描述整个项目通过Maven构建，大致结构如下： 核心Spring框架一个module sprin… 博文 来自： 开…

图像物体检测识别中的LBP特征 阅读数 1万+

1996年，Ojala老大爷搞出了LBP特征，也即参考文献1。当时好像并未引发什么波澜。到了… 博文 来自： 雨石

Cocos2d-x 2.2.3 使用NDK配置编译环境 阅读数 2万+

Cocos2d-x 2.2.3 使用NDK配置编译环境2014年6月11日 Cocos2d-x 3.0以下的开发环境的… 博文 来自： 巫…

强连通分量及缩点tarjan算法解析 阅读数 58万+

强连通分量： 简言之 就是找环（每条边只走一次，两两可达） 孤立的一个点也是一个连通… 博文 来自： 九…

YOLOv2训练自己的数据集（VOC格式） 阅读数 4万+

最近在用yolo来做视频中的人员检测，选择YOLO是从速度考虑，在训练数据集的过程中碰… 博文 来自： ch_l…

10个有趣的 Linux 命令 阅读数 489

在终端工作是一件很有趣的事情。今天，我们将会列举一些有趣得为你带来欢笑的Linux命… 博文

用C语言实现有限状态机--读《C专家编程》 阅读数 1万+

有限状态机(finite state machine)是一个数学概念，如果把它运用于程序中，可以发挥… 博文 来自： Sta…

掌柜的——写在七夕 阅读数 1458

记忆里，奶奶有生之年从未和爷爷红过脸。奶奶戏称自己是饲养员，爷爷也很幸福的当一头… 博文 来自： 花…