Franson



## Java中Lambda表达式的使用

**简介**

(**译者注**:虽然看着很先进，其实Lambda表达式的本质只是一个"**语法糖**",由编译器推断并帮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黑客写的代码一样,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表达式就和方法一样,它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.util.stream 包。 流(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(Generics)和注解(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NetBeans和IntelliJ IDEA 一类的工具和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; }**

昵称: Franson
园龄: 7年10个月
粉丝: 48
关注: 2
+加关注

< 2018年9月 >

| 日 | 一 | 二 | 三 | 四 | 五 | 六 |
|---|---|---|---|---|---|---|
| 26 | 27 | 28 | 29 | 30 | 31 | 1 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 30 | 1 | 2 | 3 | 4 | 5 | 6 |

搜索

找找看

谷歌搜索

常用链接

我的随笔

下面是Java lambda表达式的简单例子:

```
// 1. 不需要参数,返回值为 5
() -> 5

// 2. 接收一个参数(数字类型),返回其2倍的值
x -> 2 * x

// 3. 接受2个参数(数字),并返回他们的差值
(x, y) -> x – y

// 4. 接收2个int型整数,返回他们的和
(int x, int y) -> x + y

// 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)
(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表达式如何影响我们编码的方式。 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
```

我的评论
我的参与
最新评论
我的标签

## 随笔分类

DB(11)
java(51)
linux(5)
Other(5)
web(38)
安全(8)
操作系统(4)
大数据(7)
设计模式(2)
算法(2)
移动开发(10)
疑难杂症(1)

## 随笔档案

2018年8月 (1)
2018年6月 (2)
2018年5月 (3)
2018年4月 (5)
2018年3月 (4)
2018年1月 (3)
2017年12月 (2)
2017年9月 (2)
2017年8月 (3)
2017年7月 (6)
2017年6月 (5)
2017年5月 (5)
2017年4月 (2)
2017年3月 (3)
2017年2月 (4)
2017年1月 (1)

```
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lambda表达式来代替。 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:



## 文章分类



## 最新评论

1. Re:Java中Lambda表达式的使用
很棒！回去练练
--yingyuyuedu
2. Re:Java中Lambda表达式的使用
真的很好，不学好对不起楼主了
--小镇天
3. Re:Java中Lambda表达式的使用
楼主用心了，良心之作，必须 学好
--狂风骤起
4. Re:Java中Lambda表达式的使用
从.net借鉴来的吧。
--于为源
5. Re:Java中Lambda表达式的使用
感谢，楼🐮用心了
--雨点的名字

## 阅读排行榜

1. java获取当前路径的几种方法(120610)

```
// 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 !");
    }
};

// 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对集合进行排序。

**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 以及最后一个字母。 和前面的示例一样,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:

2. Java中Lambda表达式的使用(88481)
3. java中获取类加载路径和项目根路径的5种方法(64988)
4. ORACLE多表关联UPDATE 语句(43615)
5. Linux中修改环境变量及生效方法(40589)

## 评论排行榜

1. java分布式服务框架Dubbo的介绍与使用(19)
2. Java中Lambda表达式的使用(10)
3. 使用bootstrapvalidator的remote验证经验(9)
4. 一个故事教你看懂什么是数字证书，它的原理是什么?它的作用是什么? (2)
5. 关于使用由CA机构(EJBCA)颁发的证书实现SLLSocket双向认证服务端报null cert chain的解决方案(1)

## 推荐排行榜

1. Java中Lambda表达式的使用(17)
2. java获取当前路径的几种方法(6)
3. 一个故事教你看懂什么是数字证书，它的原理是什么?它的作用是什么? (5)
4. java中Log4J的使用笔记(3)
5. 使用bootstrapvalidator的remote验证经验(3)

```
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) ->
    ( s1.substring(s1.indexOf(" ")).compareTo( s2.substring(s2.indexOf(" ")) ) );
Arrays.sort(players, sortBySurname);

// 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.length() - 1)));
```

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, min, max, sum, 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。 在接下来的例子中,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,将用于进一步流操作。 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);
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:

```
System.out.println("下面是月薪超过 $1,400 的PHP程序员:")
phpProgrammers.stream()
          .filter((p) -> (p.getSalary() > 1400))
          .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:

```
// 定义 filters
Predicate<Person> ageFilter = (p) -> (p.getAge() > 25);
Predicate<Person> salaryFilter = (p) -> (p.getSalary() > 1400);
Predicate<Person> genderFilter = (p) -> ("female".equals(p.getGender()));

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:");
phpProgrammers.stream()
          .filter(ageFilter)
          .filter(salaryFilter)
          .filter(genderFilter)
          .forEach((p) -> System.out.printf("%s %s; ", p.getFirstName(), p.getLastName()));

// 重用filters
System.out.println("年龄大于 24岁的女性 Java programmers:");
javaProgrammers.stream()
          .filter(ageFilter)
          .filter(genderFilter)
          .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:

```
System.out.println("最前面的3个 Java programmers:");
javaProgrammers.stream()
          .limit(3)
          .forEach((p) -> System.out.printf("%s %s; ", p.getFirstName(), p.getLastName()));
```

```
System.out.println("最前面的3个女性 Java programmers:");
javaProgrammers.stream()
          .filter(genderFilter)
          .limit(3)
          .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list中,然后显示列表:

```
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.getLastName()));

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.getLastName()));
```

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.getSalary())

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(),
person.getSalary())
```

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到一个字符串,一个 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:");
TreeSet<String> javaDevLastName = javaProgrammers
          .stream()
          .map(Person::getLastName)
          .collect(toCollection(TreeSet::new));
```

Streams 还可以是并行的(parallel)。 示例如下:

```
System.out.println("计算付给 Java programmers 的所有money:");
int totalSalary = javaProgrammers
          .parallelStream()
          .mapToInt(p -> p.getSalary())
          .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, getMin, getSum或getAverage:

```
//计算 count, min, max, sum, and average for numbers
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
IntSummaryStatistics stats = numbers
```

```
            .stream()
            .mapToInt((x) -> x)
            .summaryStatistics();

System.out.println("List中最大的数字 : " + stats.getMax());
System.out.println("List中最小的数字 : " + stats.getMin());
System.out.println("所有数字的总和   : " + stats.getSum());
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,我们还学习了如何使用lambda表达式与Comparator 类来对Java集合进行排序。

分类: java





« 上一篇: 设计模式之---模板方法template method的使用
» 下一篇: Spring MVC详细示例实战教程【转】



**评论**

#1楼 2017-08-01 09:16 | Cloud_strife

简洁明了，非常好!

支持(0) 反对(0)

#2楼 2017-09-30 14:43 | 混天绫

很好，有收获

支持(0) 反对(0)

#3楼 2018-04-08 18:00 | leeib

java8系列文章：http://www.hao124.net/article/86
lambda表达式的组成及使用：http://www.hao124.net/article/89

支持(1) 反对(0)

#4楼 2018-07-11 14:49 | JoneZP

lambda表达式常用的都写出来了，写的贼好。

支持(0) 反对(0)

#5楼 2018-07-22 22:10 | 邪恶的无悔

66666

支持(0) 反对(0)

#6楼 2018-07-30 13:55 | 雨点的名字

感谢，楼🐱用心了

支持(0) 反对(0)

#7楼 2018-07-31 10:01 | 于为源

从.net借鉴来的吧。

支持(0) 反对(0)

#8楼 2018-07-31 15:36 | 狂风骤起

楼主用心了，良心之作，必须 学好

支持(0) 反对(0)

#9楼 2018-08-07 11:25 | 小镇天

真的很好，不学好对不起楼主了

支持(0) 反对(0)

#10楼 2018-08-21 08:55 | yingyuyuedu

很棒！回去练练

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

**注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。**