[Java Lambda表达式入门](http://blog.csdn.net/renfufei/article/details/24600507)

标签： [java](http://www.csdn.net/tag/java)[java8](http://www.csdn.net/tag/java8)[lambda](http://www.csdn.net/tag/lambda)[stream](http://www.csdn.net/tag/stream)[Lambda表达式](http://www.csdn.net/tag/Lambda%e8%a1%a8%e8%be%be%e5%bc%8f)

2014-04-27 21:17 59335人阅读 [评论](http://blog.csdn.net/renfufei/article/details/24600507#comments)(19) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/renfufei/article/details/24600507#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Java基础（43） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

原文链接: [**Start Using Java Lambda Expressions**](http://www.developer.com/java/start-using-java-lambda-expressions.html)

下载示例程序 [**Examples.zip**](http://www.developer.com/imagesvr_ce/8451/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**](http://blog.csdn.net/renfufei) **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**](http://zh.wikipedia.org/wiki/%E8%AF%AD%E6%B3%95%E7%B3%96)",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)**](http://www.oracle.com/technetwork/java/javase/downloads/index.html): 从Oracle下载Java 8,也可以和NetBeans IDE一起下载  
[**NetBeans IDE 8**](https://netbeans.org/downloads/index.html): 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:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

1. // 1. 不需要参数,返回值为 5
2. () -> 5
4. // 2. 接收一个参数(数字类型),返回其2倍的值
5. x -> 2 \* x
7. // 3. 接受2个参数(数字),并返回他们的差值
8. (x, y) -> x – y
10. // 4. 接收2个int型整数,返回他们的和
11. (**int** x, **int** y) -> x + y
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:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

1. String[] atp = {"Rafael Nadal", "Novak Djokovic",
2. "Stanislas Wawrinka",
3. "David Ferrer","Roger Federer",
4. "Andy Murray","Tomas Berdych",
5. "Juan Martin Del Potro"};
6. List<String> players =  Arrays.asList(atp);
8. // 以前的循环方式
9. **for** (String player : players) {
10. System.out.print(player + "; ");
11. }
13. // 使用 lambda 表达式以及函数操作(functional operation)
14. players.forEach((player) -> System.out.print(player + "; "));
16. // 在 Java 8 中使用双冒号操作符(double colon operator)
17. 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:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

1. // 使用匿名内部类
2. btn.setOnAction(**new** EventHandler<ActionEvent>() {
3. @Override
4. **public** **void** handle(ActionEvent event) {
5. System.out.println("Hello World!");
6. }
7. });
9. // 或者使用 lambda expression
10. btn.setOnAction(event -> System.out.println("Hello World!"));

下面是使用lambdas 来实现 Runnable接口 的示例:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

1. // 1.1使用匿名内部类
2. **new** Thread(**new** Runnable() {
3. @Override
4. **public** **void** run() {
5. System.out.println("Hello world !");
6. }
7. }).start();
9. // 1.2使用 lambda expression
10. **new** Thread(() -> System.out.println("Hello world !")).start();
12. // 2.1使用匿名内部类
13. Runnable race1 = **new** Runnable() {
14. @Override
15. **public** **void** run() {
16. System.out.println("Hello world !");
17. }
18. };
20. // 2.2使用 lambda expression
21. Runnable race2 = () -> System.out.println("Hello world !");
23. // 直接调用 run 方法(没开新线程哦!)
24. race1.run();
25. 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:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

1. String[] players = {"Rafael Nadal", "Novak Djokovic",
2. "Stanislas Wawrinka", "David Ferrer",
3. "Roger Federer", "Andy Murray",
4. "Tomas Berdych", "Juan Martin Del Potro",
5. "Richard Gasquet", "John Isner"};
7. // 1.1 使用匿名内部类根据 name 排序 players
8. Arrays.sort(players, **new** Comparator<String>() {
9. @Override
10. **public** **int** compare(String s1, String s2) {
11. **return** (s1.compareTo(s2));
12. }
13. });

使用lambdas,可以通过下面的代码实现同样的功能:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

1. // 1.2 使用 lambda expression 排序 players
2. Comparator<String> sortByName = (String s1, String s2) -> (s1.compareTo(s2));
3. Arrays.sort(players, sortByName);
5. // 1.3 也可以采用如下形式:
6. 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 :

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

1. // 1.1 使用匿名内部类根据 surname 排序 players
2. Arrays.sort(players, **new** Comparator<String>() {
3. @Override
4. **public** **int** compare(String s1, String s2) {
5. **return** (s1.substring(s1.indexOf(" ")).compareTo(s2.substring(s2.indexOf(" "))));
6. }
7. });
9. // 1.2 使用 lambda expression 排序,根据 surname
10. Comparator<String> sortBySurname = (String s1, String s2) ->
11. ( s1.substring(s1.indexOf(" ")).compareTo( s2.substring(s2.indexOf(" ")) ) );
12. Arrays.sort(players, sortBySurname);
14. // 1.3 或者这样,怀疑原作者是不是想错了,括号好多...
15. Arrays.sort(players, (String s1, String s2) ->
16. ( s1.substring(s1.indexOf(" ")).compareTo( s2.substring(s2.indexOf(" ")) ) )
17. );
19. // 2.1 使用匿名内部类根据 name lenght 排序 players
20. Arrays.sort(players, **new** Comparator<String>() {
21. @Override
22. **public** **int** compare(String s1, String s2) {
23. **return** (s1.length() - s2.length());
24. }
25. });
27. // 2.2 使用 lambda expression 排序,根据 name lenght
28. Comparator<String> sortByNameLenght = (String s1, String s2) -> (s1.length() - s2.length());
29. Arrays.sort(players, sortByNameLenght);
31. // 2.3 or this
32. Arrays.sort(players, (String s1, String s2) -> (s1.length() - s2.length()));
34. // 3.1 使用匿名内部类排序 players, 根据最后一个字母
35. Arrays.sort(players, **new** Comparator<String>() {
36. @Override
37. **public** **int** compare(String s1, String s2) {
38. **return** (s1.charAt(s1.length() - 1) - s2.charAt(s2.length() - 1));
39. }
40. });
42. // 3.2 使用 lambda expression 排序,根据最后一个字母
43. Comparator<String> sortByLastLetter =
44. (String s1, String s2) ->
45. (s1.charAt(s1.length() - 1) - s2.charAt(s2.length() - 1));
46. Arrays.sort(players, sortByLastLetter);
48. // 3.3 or this
49. 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:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

1. **public** **class** Person {
3. **private** String firstName, lastName, job, gender;
4. **private** **int** salary, age;
6. **public** Person(String firstName, String lastName, String job,
7. String gender, **int** age, **int** salary)       {
8. **this**.firstName = firstName;
9. **this**.lastName = lastName;
10. **this**.gender = gender;
11. **this**.age = age;
12. **this**.job = job;
13. **this**.salary = salary;
14. }
15. // Getter and Setter
16. // . . . . .
17. }

接下来,我们将创建两个list,都用来存放Person对象:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

1. List<Person> javaProgrammers = **new** ArrayList<Person>() {
2. {
3. add(**new** Person("Elsdon", "Jaycob", "Java programmer", "male", 43, 2000));
4. add(**new** Person("Tamsen", "Brittany", "Java programmer", "female", 23, 1500));
5. add(**new** Person("Floyd", "Donny", "Java programmer", "male", 33, 1800));
6. add(**new** Person("Sindy", "Jonie", "Java programmer", "female", 32, 1600));
7. add(**new** Person("Vere", "Hervey", "Java programmer", "male", 22, 1200));
8. add(**new** Person("Maude", "Jaimie", "Java programmer", "female", 27, 1900));
9. add(**new** Person("Shawn", "Randall", "Java programmer", "male", 30, 2300));
10. add(**new** Person("Jayden", "Corrina", "Java programmer", "female", 35, 1700));
11. add(**new** Person("Palmer", "Dene", "Java programmer", "male", 33, 2000));
12. add(**new** Person("Addison", "Pam", "Java programmer", "female", 34, 1300));
13. }
14. };
16. List<Person> phpProgrammers = **new** ArrayList<Person>() {
17. {
18. add(**new** Person("Jarrod", "Pace", "PHP programmer", "male", 34, 1550));
19. add(**new** Person("Clarette", "Cicely", "PHP programmer", "female", 23, 1200));
20. add(**new** Person("Victor", "Channing", "PHP programmer", "male", 32, 1600));
21. add(**new** Person("Tori", "Sheryl", "PHP programmer", "female", 21, 1000));
22. add(**new** Person("Osborne", "Shad", "PHP programmer", "male", 32, 1100));
23. add(**new** Person("Rosalind", "Layla", "PHP programmer", "female", 25, 1300));
24. add(**new** Person("Fraser", "Hewie", "PHP programmer", "male", 36, 1100));
25. add(**new** Person("Quinn", "Tamara", "PHP programmer", "female", 21, 1000));
26. add(**new** Person("Alvin", "Lance", "PHP programmer", "male", 38, 1600));
27. add(**new** Person("Evonne", "Shari", "PHP programmer", "female", 40, 1800));
28. }
29. };

现在我们使用forEach方法来迭代输出上述列表:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

1. System.out.println("所有程序员的姓名:");
2. javaProgrammers.forEach((p) -> System.out.printf("%s %s; ", p.getFirstName(), p.getLastName()));
3. 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%:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

1. System.out.println("给程序员加薪 5% :");
2. Consumer<Person> giveRaise = e -> e.setSalary(e.getSalary() / 100 \* 5 + e.getSalary());
4. 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:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

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:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

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()));
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()));
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:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

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()));

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list中,然后显示列表:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

1. System.out.println("根据 name 排序,并显示前5个 Java programmers:");
2. List<Person> sortedJavaProgrammers = javaProgrammers
3. .stream()
4. .sorted((p, p2) -> (p.getFirstName().compareTo(p2.getFirstName())))
5. .limit(5)
6. .collect(toList());
8. sortedJavaProgrammers.forEach((p) -> System.out.printf("%s %s; %n", p.getFirstName(), p.getLastName()));
10. System.out.println("根据 salary 排序 Java programmers:");
11. sortedJavaProgrammers = javaProgrammers
12. .stream()
13. .sorted( (p, p2) -> (p.getSalary() - p2.getSalary()) )
14. .collect( toList() );
16. 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:

**[plain]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

1. System.out.println("工资最低的 Java programmer:");
2. Person pers = javaProgrammers
3. .stream()
4. .min((p1, p2) -> (p1.getSalary() - p2.getSalary()))
5. .get()
7. System.out.printf("Name: %s %s; Salary: $%,d.", pers.getFirstName(), pers.getLastName(), pers.getSalary())
9. System.out.println("工资最高的 Java programmer:");
10. Person person = javaProgrammers
11. .stream()
12. .max((p, p2) -> (p.getSalary() - p2.getSalary()))
13. .get()
15. 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:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

1. System.out.println("将 PHP programmers 的 first name 拼接成字符串:");
2. String phpDevelopers = phpProgrammers
3. .stream()
4. .map(Person::getFirstName)
5. .collect(joining(" ; ")); // 在进一步的操作中可以作为标记(token)
7. System.out.println("将 Java programmers 的 first name 存放到 Set:");
8. Set<String> javaDevFirstName = javaProgrammers
9. .stream()
10. .map(Person::getFirstName)
11. .collect(toSet());
13. System.out.println("将 Java programmers 的 first name 存放到 TreeSet:");
14. TreeSet<String> javaDevLastName = javaProgrammers
15. .stream()
16. .map(Person::getLastName)
17. .collect(toCollection(TreeSet::**new**));

Streams 还可以是并行的(parallel)。 示例如下:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

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, getMin, getSum或getAverage:

**[java]** [view plain](http://blog.csdn.net/renfufei/article/details/24600507) [copy](http://blog.csdn.net/renfufei/article/details/24600507)

[在CODE上查看代码片](https://code.csdn.net/snippets/315636)

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();
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());

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

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。