技术分享 > Java >

Java 8 的 lambda 表达式

58 回答

5

收藏(87)

Java 8 预计将在 2013 年发布，Java 8 将支持 Lambda 功能，尽管该规范还在不断的变化，但是 Java 8 的开发版已经实现了对 lambda 的支持。

关于 lambda 表达式的定义请看维基百科。

该文章将带你熟悉 lambda 语法，以及使用集合 API 中的 lambda 以及相关的语言增强，本文所有的代码都是在 JDK 8 lambda build b39 编译。

功能接口

只包含一个方法的接口被称为功能接口，Lambda 表达式用用于任何功能接口适用的地方。

`java.awt.event.ActionListener` 就是一个功能接口，因为它只有一个方法：`void actionPerformed(ActionEvent)`. 在 Java 7 中我们会编写如下代码：

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        ui.dazzle(e.getModifiers());
    }
});
```

而 Java 8 中可以简化为：

```
button.addActionListener(e -> { ui.dazzle(e.getModifiers()); });
```

编译器知道lambda 表达式必须符合 `void actionPerformed(ActionEvent)` 方法的定义。看起来 lambda 实体返回 void，实际上它可以推断出参数 e 的类型是 `java.awt.event.ActionEvent`.

函数集合

Java 8 的类库包含一个新的包 `java.util.functions` ，这个包中有很多新的功能接口，这些接口可与集合 API 一起使用。

java.util.functions.Predicate

使用谓词 (Predicate) 来筛选集合：

```
List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "Dave");
List<String> filteredNames = names
        .filter(e -> e.length() >= 4)
        .into(new ArrayList<String>());
for (String name : filteredNames) {
    System.out.println(name);
}
```

这里我们有两个新方法：

- `Iterable<T> filter(Predicate<? super T>)` 用于获取元素满足某个谓词返回 true 的结果
- `<A extends Fillable<? super T>> A into(A)` 将用返回的结果填充 ArrayList

java.util.functions.Block

我们可使用一个新的迭代器方法来替换 for 循环 `void forEach(Block<? super T>)`:

```
List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "Dave");
names
    .filter(e -> e.length() >= 4)
    .forEach(e -> { System.out.println(e); });
```

`forEach()` 方法是 internal iteration 的一个实例：迭代过程在 `Iterable` 和 `Block` 内部进行，每次可访问一个元素。

最后的结果就是用更少的代码来处理集合：

```
List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "Dave");
names
    .mapped(e -> { return e.length(); })
    .asIterable() // returns an Iterable of BiValue elements
                  // an element's key is the person's name, its value is the stri
    .filter(e -> e.getValue() >= 4)
    .sorted((a, b) -> a.getValue() - b.getValue())
    .forEach(e -> { System.out.println(e.getKey() + '\t' + e.getValue()); });
```

这样做的优点是：

元素在需要的时候才进行计算
    如果我们取一个上千个元素的集合的前三条时，其他元素就不会被映射
鼓励使用方法链
    我们无需才存储中间结果来构建新的集合
内部迭代过程因此大多数细节
    例如，我们可以通过下面代码来并行 map() 操作
    writing `myCollection.parallel().map(e -> e.length())`.

方法引用

我们可通过 :: 语法来引用某个方法。方法引用被认为是跟 lambda 表达式一样的，可用于功能接口所适用的地方。

我们可以引用一个静态方法：

```
executorService.submit(MethodReference::sayHello);

private static void sayHello() {
        System.out.println("hello");
}
```

或者是一个实例的方法：

```
Arrays.asList("Alice", "Bob", "Charlie", "Dave").forEach(System.out::println);
```

我们也可以创建工程方法并将构造器引用赋值给 `java.util.functions.Factory`:

```
Factory<Biscuit> biscuitFactory = Biscuit::new;
Biscuit biscuit = biscuitFactory.make();
```

最后，我们创建一个引用到随意实例的例子：

```
interface Accessor<BEAN, PROPERTY> {
        PROPERTY access(BEAN bean);
}

public static void main(String[] args) {
        Address address = new Address("29 Acacia Road", "Tunbridge Wells");
        Accessor<Address, String> accessor = Address::getCity;
        System.out.println(accessor.access(address));
}
```

这里我们无需绑定方法引用到某个实例，我们直接将实例做为功能接口的参数进行传递。

默认方法

直到今天的 Java ，都不可能为一个接口添加方法而不会影响到已有的实现类。而 Java 8 允许你为接口自身指定一个默认的实现：

```
interface Queue {
        Message read();
        void delete(Message message);
        void deleteAll() default {
                Message message;
                while ((message = read()) != null) {
                        delete(message);
                }
        }
}
```

子接口可以覆盖默认的方法：

```
interface BatchQueue extends Queue {
        void setBatchSize(int batchSize);
        void deleteAll() default {
                setBatchSize(100);
                Queue.super.deleteAll();
        }
}
```

或者子接口也可以通过重新声明一个没有方法体的方法来删除默认的方法：

```
interface FastQueue extends Queue {
        void deleteAll();
}
```

这个将强制所有实现了 FastQueue 的类必须实现 deleteAll() 方法。

HotSpot 实现

lambda 不只是可以减少很多代码的编写，其字节码和运行时的实现也比 Java 7 中的匿名类的效率更高。针对每一个 lambda 表达式，编译器都会创建一个对应的形如 lambda$1() 这样的方法。这个过程被称之为 *lambda body desugaring*. 当遇见一个 lambda 表达式，编译器将会发起一个 `invokedynamic` 调用，并从目标功能接口中获取返回值。

深入阅读

本文很多内容都基于 Brian Goetz 的文章：State of the Lambda, State of the Lambda: Libraries Edition and Translation of Lambda Expressions. 这些文字详细描述了 lambda 语法、变量捕获、类型接口和编译等内容。

英文原文，OSCHINA原创翻译

Java lambda 精华 OSCHINA原创翻译

举报 分享


红薯
发帖于5年前 58回/40K+阅

共有58个评论 最后回答: 8个月前 按默认排序 显示最新评论

0
哟西
评论(0) 引用此评论 举报


IdleMan
5年前

0
默认方法这个不错哦：）
--- 共有 3 条评论 ---
爱国者： 这些所谓的新特性，scala已经提供了 5年前


红薯
5年前

dytes： 终于改进了。 默认方法这个好像F#接口里面也有这个功能。 5年前
晕dows： +1 5年前

评论(3) 引用此评论 举报

0

大神级的人物。

徐建兴
5年前

评论(0) 引用此评论 举报

0

搜噶！

Beyond...
5年前

评论(0) 引用此评论 举报

0

看不懂，还是Java6好。

微信流量交易张子游
5年前

--- 共有 1 条评论 ---

梅公子： 我也卡不懂介个 5年前

评论(1) 引用此评论 举报

0

默认方法，这个原理是怎么实现的呢？

zm112358
5年前

评论(0) 引用此评论 举报

0

lambda没有c#看起来简洁。。

Evo
5年前

评论(0) 引用此评论 举报

0

Java: "C#有的我都要有 :-)".

all_br...
5年前

--- 共有 3 条评论 ---

dytes： @all_bright 虽然没太多更新，但稳定也算一个有点吧，一直跟着微软跑，也挺累的。 4年前
all_bright： 回复 @dytes : Java语言本身已无长处 :-) 5年前
dytes： 互相学习长处么，没必要害臊，哈哈！ 5年前

评论(3) 引用此评论 举报

0

语法也变得有意思起来了

kiddkai
5年前

评论(0) 引用此评论 举报

0

看着不错。不过我们这边现在开发和部署还都是在用5，用上8，估计得10年之后了 。

hunterli
5年前

评论(0) 引用此评论 举报

1 2 3 4 5 6 下一页

Format

提交回答