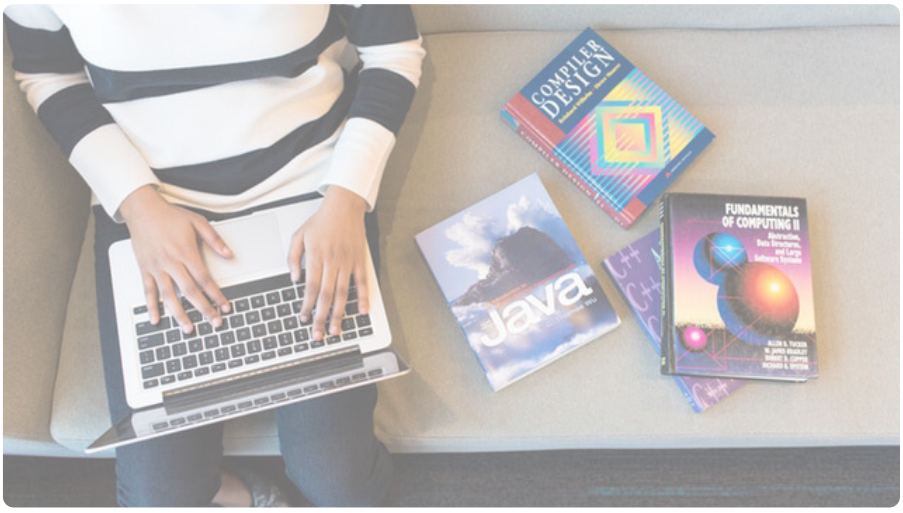
13 差异对比：集合在 Java 7 和 8 有何不同和改进

更新时间：2019-09-19 09:39:16



> 时间像海绵里的水，只要你愿意挤，总还是有的。
>
> ——鲁迅

引导语

Java 8 在 Java 7 的基础上，做了一些改进和优化，但我们在平时工作中，或者直接升级到 Java 8 的过程中，我们好像无需做任何兼容逻辑，那么 Java 8 底层是如何处理的呢，在改进的同时，是如何优雅兼容 Java 老版本，让使用者无需感知，接下来我们通过对比 Java 7 和 8 的差异，来展示 Java 8 是如何优雅升级的。

1 通用区别

1.1 所有集合都新增了forEach 方法

List、Set、Map 在 Java 8 版本中都增加了 forEach 的方法，方法的入参是 Consumer，Consumer 是一个函数式接口，可以简单理解成允许一个入参，但没有返回值的函数式接口，我们以 ArrayList 的 forEach 的源码为例，来看下方法是如何实现的 ：

```
@Override
public void forEach(Consumer<? super E> action) {
  // 判断非空
  Objects.requireNonNull(action);
  // modCount的原始值被拷贝
  final int expectedModCount = modCount;
  final E[] elementData = (E[]) this.elementData;
  final int size = this.size;
  // 每次循环都会判断数组有没有被修改，一旦被修改，停止循环
  for (int i=0; modCount == expectedModCount && i < size; i++) {
    // 执行循环内容，action 代表我们要干的事情
    action.accept(elementData[i]);
```

```
    throw new ConcurrentModificationException();
  }
}
```

从这段源码中，很容易产生两个问题：

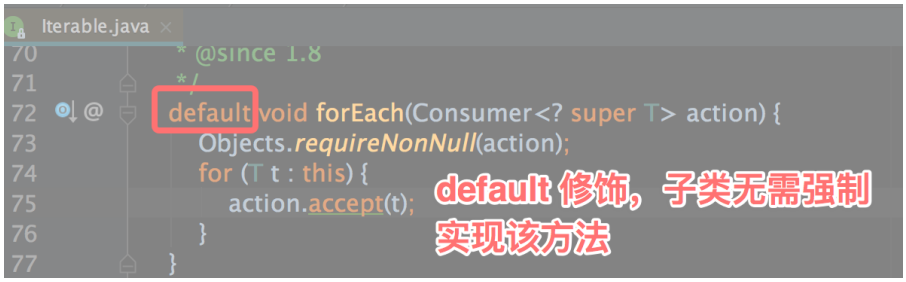
1、action.accept 到底是个啥？

action.accept 就是你在 for 循环中要干的事情，你可以进行任何事情，比如我们打印一句话，如下：

```
public void testForEach(){
  List<Integer> list = new ArrayList<Integer>(){{
    add(1);
    add(3);
    add(2);
    add(4);
  }};
  // value 是每次循环的入参，就是 list 中的每个元素
  list.forEach( value->log.info("当前值为：{}",value));
}
输出为：
当前值为：1
当前值为：3
当前值为：2
当前值为：4
```


log.info("当前值为：{}",value)就是我们要干的事情，就是 action。

2.、forEach 方法上打了 @Override 注解，说明该方法是被继承实现的，该方法是被定义在 Iterable 接口上的，Java 7 和 8 的 ArrayList 都实现了该接口，但我们在 Java 7 的 ArrayList 并没有发现有实现该方法，编译器也木有报错，这个主要是因为 Iterable 接口的 forEach 方法被加上了 default 关键字，这个关键字只会出现在接口类中，被该关键字修饰的方法无需强制要求子类继承，但需要自己实现默认实现，我们看下源码：



不仅仅是 forEach 这一个方法是这么干的，List、Set、Map 接口中很多新增的方法都是这么干的，通过 default 关键字，可以让 Java 7 的集合子类无需实现 Java 8 中新增的方法。

如果想在接口中新增一个方法，但又不想子类强制实现该方法时，可以给该方法加上 default 关键字，这个在实际工作中，也经常使用到，算是重构的小技巧吧。

1.2 List 区别

1.2.1 ArrayList

List 其它方面 java7 和 8 并没有改动。

1.3 Map 区别

1.3.1 HashMap

1. 和 ArrayList 一样，Java 8 中 HashMap 在无参构造器中，丢弃了 Java 7 中直接把数组初始化 16 的做法，而是采用在第一次新增的时候，才开始扩容数组大小；
2. hash 算法计算公式不同，Java 8 的 hash 算法更加简单，代码更加简洁；
3. Java 8 的 HashMap 增加了红黑树的数据结构，这个是 Java 7 中没有的，Java 7 只有数组 + 链表的结构，Java 8 中提出了数组 + 链表 + 红黑树的结构，一般 key 是 Java 的 API 时，比如说 String 这些 hashcode 实现很好的 API，很少出现链表转化成红黑树的情况，因为 String 这些 API 的 hash 算法够好了，只有当 key 是我们自定义的类，而且我们覆写的 hashcode 算法非常糟糕时，才会真正使用到红黑树，提高我们的检索速度。

也是因为 Java 8 新增了红黑树，所以几乎所有操作数组的方法的实现，都发生了变动，比如说 put、remove 等操作，可以说 Java 8 的 HashMap 几乎重写了一遍，所以 Java 7 的很多问题都被 Java 8 解决了，比如扩容时极小概率死锁，丢失数据等等。

4. 新增了一些好用的方法，比如 getOrDefault，我们看下源码，非常简单：


```
// 如果 key 对应的值不存在，返回期望的默认值 defaultValue
public V getOrDefault(Object key, V defaultValue) {
    Node<K,V> e;
    return (e = getNode(hash(key), key)) == null ? defaultValue : e.value;
}
```

还有 putIfAbsent(K key, V value) 方法，意思是，如果 map 中存在 key 了，那么 value 就不会覆盖，如果不存在 key ，新增成功。

还有 compute 方法，意思是允许我们把 key 和 value 的值进行计算后，再 put 到 map 中，为防止 key 值不存在造成未知错误，map 还提供了 computeIfPresent 方法，表示只有在 key 存在的时候，才执行计算，demo 如下：

```
@Test
public void compute(){
  HashMap<Integer,Integer> map = Maps.newHashMap();
  map.put(10,10);
  log.info("compute 之前值为： {}",map.get(10));
  map.compute(10,(key,value) -> key * value);
  log.info("compute 之后值为： {}",map.get(10));
  // 还原测试值
  map.put(10,10);

  // 如果为 11 的 key 不存在的话，需要注意 value 为空的情况，下面这行代码就会报空指针
  //  map.compute(11,(key,value) -> key * value);

  // 为了防止 key 不存在时导致的未知异常，我们一般有两种办法
  // 1： 自己判断空指针
  map.compute(11,(key,value) -> null == value ? null : key * value);
```

目录

```
}
结果是：
compute 之前值为：10
compute 之后值为：100
computeIfPresent 之后值为：null（这个结果中，可以看出，使用 computeIfPresent 避免了空指针
```

上述 Java 8 新增的几种方法非常好用，在实际工作中，可以大大减少我们的代码量，computeIfPresent 的源码就不贴了，有兴趣可以去 github 上面查看，主要的实现原理如下：

- 找到 key 对应的老值，会分别从数组、链表、红黑树中找；
- 根据 key 和老值进行计算，得到新值；
- 用新值替换掉老值，可能是普通替换、链表替换或红黑树替换。

1.3.2 LinkedHashMap

由于 Java 8 的底层数据有变动，导致 HashMap 操作数据的方法几乎重写，也使 LinkedHashMap 的实现名称上有所差异，原理上都相同，我们看下面的图，左边是 Java 7，右边是 Java 8。

从图中，我们发现 LinkedHashMap 的方法名有所修改，底层的实现逻辑其实都差不多的。

1.4 其他区别

1.4.1 Arrays 提供了很多 parallel 开头的方法。

Java 8 的 Arrays 提供了一些 parallel 开头的方法，这些方法支持并行的计算，在数据量大的时候，会充分利用 CPU ，提高计算效率，比如说 parallelSort 方法，方法底层有判断，只有数据量大于 8192 时，才会真正走并行的实现，在实际的实验中，并行计算的确能够快速的提高计算速度。

1.5 面试题

1. Java 8 在 List、Map 接口上新增了很多方法，为什么 Java 7 中这些接口的实现者不需要强制实现这些方法呢？

答：主要是因为这些新增的方法被 default 关键字修饰了，default 一旦修饰接口上的方法，我们需要在接口的方法中写默认实现，并且子类无需强制实现这些方法，所以 Java 7 接口的实现者无需感知。

答：有的，比如说 getOrDefault、putIfAbsent、computeIfPresent 方法等等，具体使用细节参考上文。

3. 说说 computeIfPresent 方法的使用姿势？

答：computeIfPresent 是可以对 key 和 value 进行计算后，把计算的结果重新赋值给 key，并且如果 key 不存在时，不会报空指针，会返回 null 值。

4. Java 8 集合新增了 forEach 方法，和普通的 for 循环有啥不同？

答：新增的 forEach 方法的入参是函数式的接口，比如说 Consumer 和 BiConsumer，这样子做的好处就是封装了 for 循环的代码，让使用者只需关注实现每次循环的业务逻辑，简化了重复的 for 循环代码，使代码更加简洁，普通的 for 循环，每次都需要写重复的 for 循环代码，forEach 把这种重复的计算逻辑吃掉了，使用起来更加方便。

5. HashMap 8 和 7 有啥区别？

答：HashMap 8 和 7 的差别太大了，新增了红黑树，修改了底层数据逻辑，修改了 hash 算法，几乎所有底层数组变动的方法都重写了一遍，可以说 Java 8 的 HashMap 几乎重新了一遍。

总结

总体来说，List 方面是小改动，HashMap 几乎重写了一遍，所有的集合都新增了函数式的方法，比如说 forEach，也新增了很多好用的函数，比如说 getOrDefault，这些函数可以大大减少我们的代码量，让我们把关注点聚焦在业务逻辑的实现上，这其实是一种思想，把繁琐重复的计算逻辑抽取出来，从计算逻辑中扩展出业务逻辑的口子，让使用者只专心关注业务逻辑的实现即可。

想要了解更多差异，也可直接前往 JDK 8 新特性查看，地址为：http://openjdk.java.net/projects/jdk8/features#103。

← 12 彰显细节：看集合源码对我们实际工作的帮助和应用

14 简化工作：Guava Lists Maps 实际工作运用和源码 →

精选留言 3

欢迎在这里发表留言，作者筛选后可公开显示

和尚码代码

太难了，看完又没了，焦急的等待

1 回复 2019-09-20

qq_Ezio_1

老师 这节看到lambda表达式，专栏后期会有java8特性讲解吗

目录

文贺 回复 **qq_Ezio_1**

有的，第八章专门说 Lambda，和大家一起如何看 Lambda 的源码和使用。

回复 2019-09-20 14:04:02

qq_Ezio_1

老师 这个log是哪个包的类。找半天找不到，专栏源码可以共享吗

0 回复 2019-09-19

文贺 回复 **qq_Ezio_1**

idea 装一个 lombok 的插件就好了。源码在第三小节有贴出来： 源码解析：https://github.com/luanqiu/java8 文章 demo：https://github.com/luanqiu/java8_demo

回复 2019-09-20 14:06:02

qq_Ezio_1 回复 **文贺**

链接打开404.。。。。

回复 2019-09-21 16:17:55

文贺 回复 **qq_Ezio_1**

没有问题的哈，麻烦你检查下网址输入是否有误哦，可以把 404 的网址贴出来哈。

回复 2019-09-22 16:43:52