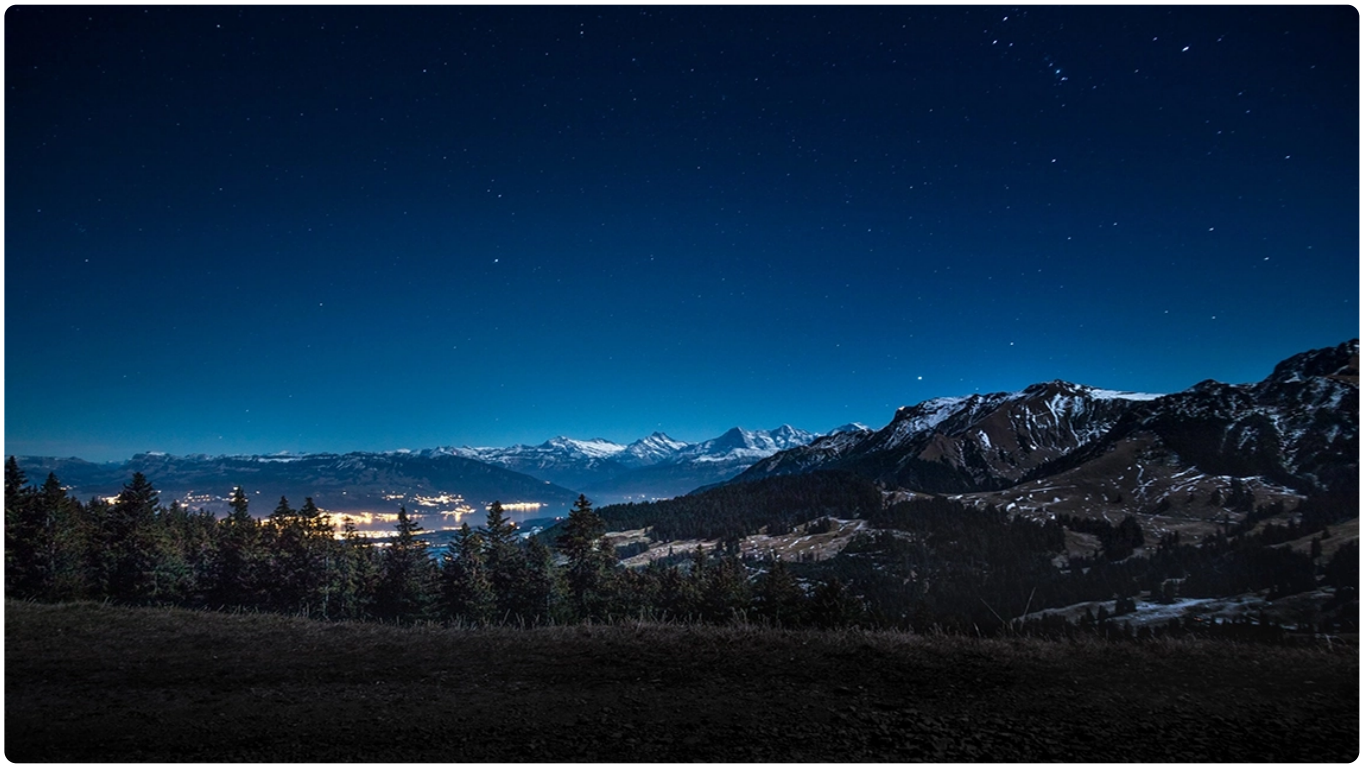
用户故事 | 保持好奇心，积极拥抱变化

2021-12-15 贾慧鑫

《深入剖析Java新特性》 课程介绍 >



讲述：梁正霖
时长 05:51 大小 5.37M

你好，我是贾慧鑫，目前是蔚来汽车的一名后端开发工程师，平时主要使用 Java 语言。

9 月份，JDK 17 正式版发布，作为 Java 的又一长期版本，最近我们的新项目也在尝试将 JDK 11 升级到 17。恰巧就在这时候，我看到了范学雷老师推出的《深入剖析 Java 新特性》，于是毫不犹疑地订阅了。

我认为，作为一名研发工程师，应该保持一颗好奇心，积极拥抱技术的变化，不断地去学习和迭代自己的知识。接下来，我想跟大家分享一下工作以来我学习和使用 Java 的一些心得体会，以及这门课程对我的帮助。

学习 Java 的心得

从 Java 8 开始，Java 语言正式引入了函数式编程。Stream 和 Lambda 的结合极大地简化了我们在 Java7 中对集合类的操作。现在再看 Java 7，会有种回不去的感觉。因为相比之下，用 Java 7 编写的代码又复杂又冗余。如果你还用过 Java 11 或者 Java 17，相信这种感觉会更强烈。

而且，Java 语言也在不断借鉴其他编程语言，让代码变得更加简洁高效。下面我通过一个例子来展示一下不同 Java 版本的变化。

复制代码

```
// 假如给定一个由数字1，2，3，4构成的List，要求把元素都值都扩大一倍

// java8
List<Integer> res = initList.stream().map(i -> i * 2).collect(Collectors.toLis

// java11
var res = initList.stream().map(i -> i * 2).collect(Collectors.toUnmodifiableL

// java17
var res = initList.stream().map(i -> i * 2).toList();
```

首先，从代码的样式来看，Java 17 的版本无疑是最简洁的。另外，Java 语言在函数式编程方面对于不可变数据的支持也更完善了。比方说，Java 8 的 Collectors.toList() 返回的只是一个可变的 List；在 Java 11 中，就新增了 Collectors.toUnmodifiableList() 方法，支持我们返回不可变的 List；而在 Java 17 中，我们直接通过 toList 就能返回不可变的 List 了。

不可变数据可以让开发更加简单、可回溯、测试友好，它减少了很多可能的副作用，也就是说，减少了 Bug 的出现。尤其是针对并发编程方面，我们应该多使用函数式编程和不可变数据来实现我们的代码逻辑，尽量避免加锁带来的负载逻辑和系统开销。

在平时的工作中，我也尽量去编写纯函数，使用不可变的数据，这样做为我消除了很多不必要的问题。我想用我写的代码来举个例子：

复制代码

```
// lombok是我们现在web开发常用的扩展包，其中setter是一个可变操作，我们可以把它在配置文件中
lombok.setter.flagUsage = error
lombok.data.flagUsage = error

```

```
// 声明
@With
@Getter
@Builder(toBuilder = true)
@AllArgsConstructor(staticName = "of")
@NoArgsConstructor(staticName = "of")
@FieldDefaults(makeFinal = true, level = AccessLevel.PRIVATE)
public class Model() {
    String id;
    String name;
    String type;
}

// 构建Model
var model_01 = Model.of("101", "model_01", "model");

// 构建空Model
var model_02 = Model.of();

// 构建指定参数的Model
var model_03 = Moder.toBuilder().id("301").name("model_03").build();

// 修改Model的一个值，通过@With来生成一个全新的model
var model_04 = model_01.withName("model_04");

// 修改多个值，通过@Builder来生成一个全新的model
var model_05 = model_01.toBuilder().name("model_05").type("new_model").build()
```

上面这段代码是我工作中常用的 Model 类，它结合 Lombok 简化了冗余代码。这段代码使用 of() 方法构建新对象，withXX() 方法修改单个参数，toBuilder() 方法修改多个参数，修改后会返回新的对象。这种代码编写方式保证了数据的不可变性，让我们的开发更加简单、可回溯、测试友好，减少了任何可能的副作用。

《深入剖析 Java 新特性》对我的帮助

从今年九月份到现在，Java17 正式发布这么短的时间内，范学雷老师就出了这门课程真的非常棒。《深入剖析 Java 新特性》不仅帮我梳理了 Java 新特性的条目和使用方法，还把为什么会有这项新特性给我们讲了讲，这点真的很重要，这让我对新特性有了更深的理解。

我之前看其他讲解封闭类的文章就一直不理解，Java 这么多年都用过来了，为什么还需要封闭类这东西呢？但范老师在关于封闭类这一讲里，就举了一个形状的案例。我才知道，原来以前类的设计要么是全开放，要么是全封闭的。它缺少中间的可控状态，这让我茅塞

顿开，对这个新特性有了新的理解。虽然这个封闭类在我编写业务代码时不常用到，但是在编写公共工具类的时候，可以增加代码的可控性，确实是一个不错的选择。

在档案类这一讲，我们看到，Java 在构建不可变对象的路上带我们又向前走了一步，进一步简化了我们的代码实现。刚才，我简单描述了一下不可变数据的优势以及结合 Lombok 的实现方式，现在有了 Record 以后，我们就可以把 Model 都交给 Record 来实现了。

我还去查询了 Lombok 的版本迭代，Lombok 在 1.18.20 版本就对 Record 进行了支持，1.18.22 版本开始全面支持 JDK17 的其他新特性，更详细的内容大家可以去 Lombok 官网看一看。这样一来，我们就可以把上面的 Model 简化为 Record 来实现：

复制代码

```
// 注意：@NoArgsConstructor,@RequiredArgsConstructor,@AllArgsConstructor 只支持c
@With
@Builder(toBuilder = true)
public record Model(String id, String name, String type) {}
```

在上面这段代码中，Record 已经极大简化了我们的代码实现。不过我个人有点小遗憾，希望 Record 能有一个默认.of() 的对象构造方法，如果能填补上这个空缺，会更符合函数式编程的风格。

另外，课程后面的模式匹配和 switch 表达式都为我们提供了更好的选择，让我们的代码更加简洁和灵活。所以如果你还停留在 Java 8 的话，还是尽快学习一下这门课程吧。

写在最后

谈到 Java 相关的函数式编程技术，首先应该感谢一下我的 mentor，他不仅是一位乐于分享知识的技术大神，还是 Kotlin 语言的布道师，也经常为开源社区做贡献。如果你对 Kotlin 语言或者函数式编程感兴趣，可以去看一下他的博客：hltj.me。

感谢极客时间给开发人员提供了这么多专业的、有价值的计算机相关技术，而且现在的新课程都结合了最新迭代的相关技术，就像范老师的这门课程，就在最短的时间内为我们带来了 Java 最新的特性分析和讲解，这对于 Java 语言从业者无疑是一份宝贵的学习资料。

也期待极客时间在未来可以有更多硬核知识分享，如果能有响应式编程和函数式编程相关的课程，我一定会第一时间购买。

期待你在评论区留言，我们一起讨论更多新特性的知识吧！

上一篇 14 | 禁止空指针，该怎么避免崩溃的空指针？

精选留言 (4)

范学雷
2021-12-15

谢谢，非常棒的分享！

展开

2

ABC
2021-12-15

看了一下源码里面,Stream.java里面封装了一层返回不可变列表的操作:

```
@SuppressWarnings("unchecked")
    default List<T> toList() {
        return (List<T>) Collections.unmodifiableList(new ArrayList<>(Arrays.asList(this.t...
```

展开

bigben
2021-12-15

比较接地气，学到就能用。

展开


aoe

2021-12-15

原来lombok还可以直接用of实例化对象！17获得不可变list相当方便啊！感谢分享！