



下载APP

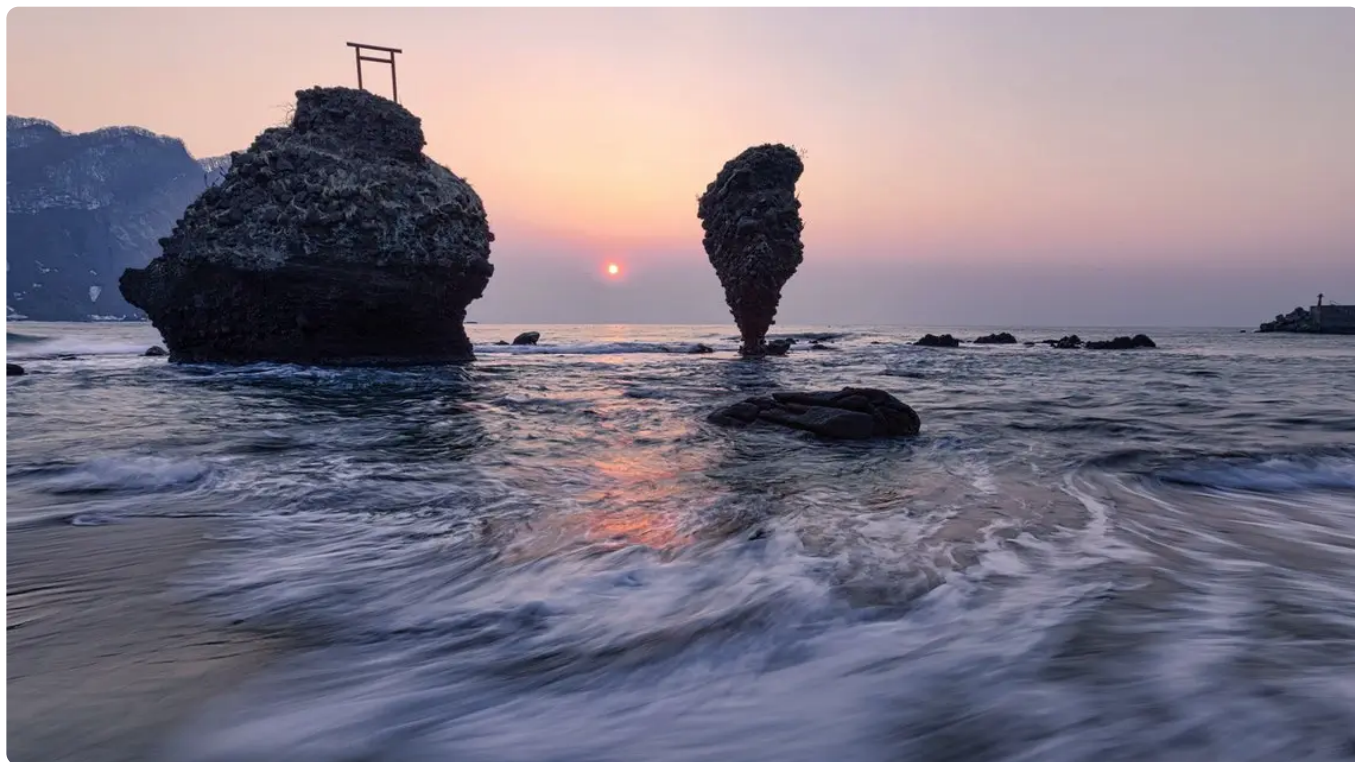


16 | 改进的废弃，怎么避免使用废弃的特性？

2021-12-22 范学雷

《深入剖析Java新特性》

课程介绍 >



讲述：范学雷

时长 08:29 大小 7.79M



你好，我是范学雷。今天，我们讨论 Java 公开接口的废弃。

像所有的事物一样，公开接口也有生命周期。要废弃那些被广泛使用的、或者还有人使用的公开接口，是一个非常痛苦的过程。该怎么废弃一个公开接口，该怎么减少废弃接口对我们的影响呢？这是这一次我们要讨论的话题。

领资料

我们先来看看阅读案例。

阅读案例



在 JDK 中，一个公开的接口，可能会因为多种多样的原因被废弃。比如说，这个接口的设计是危险的，或者有了更新的、更好的替代接口。不管是什么原因，废弃接口的使用者们

都需要尽快迁移代码，转换到替代方案上来。

在 JDK 中，公开接口的废弃需要使用两种不同的机制，也就是“Deprecated” 注解（annotation）和“Deprecated” 文档标记（JavaDoc tag）。

Deprecated 的注解会编译到类文件里，并且可以在运行时查验。这就允许像 javac 这样的工具检测和标记已废弃接口的使用情况了。

Deprecated 文档标记用于描述废弃接口的文档中。除了标记接口的废弃状态之外，一般情况下，我们还要描述废弃的原因和替代的方案。

下面的这段代码，就是使用 Java 注解和文档标记来废弃一个公开接口的例子。

[复制代码](#)

```
1 public sealed abstract class Digest {
2     /**
3      * -- snipped
4      *
5      * @deprecated This method is not performance friendly. Use
6      *              {@link #digest(byte[], byte[]) instead.
7      */
8     @Deprecated
9     public abstract byte[] digest(byte[] message);
10
11     // snipped
12     public void digest(byte[] message, byte[] digestValue) {
13         // snipped
14     }
15 }
```

如果一段程序使用了废弃接口，编译的时候，就会提出警告。但是，有很多编译环境的配置，把编译警告看作是编译错误。为了解决这样的问题，JDK 还提供了“消除使用废弃接口的编译警告”的选项。也就是 SuppressWarnings 注解。

[复制代码](#)

```
1 @SuppressWarnings("deprecation")
2 public static void main(String[] args) {
3     try {
4         Digest.of("SHA-256")
5             .digest("Hello, world!".getBytes());
6     }
```

```
6     } catch (NoSuchAlgorithmException ex) {  
7         // ignore  
8     }  
9 }
```

公开接口的废弃机制，是在 JDK 1.5 的时候发布的。这种机制像一座设计者和使用者之间的沟通桥梁，减轻了双方定义或者使用废弃接口的痛苦。

遗憾的是，直到现在，公开接口的废弃，依然是一个复杂、痛苦的过程。一个公开的接口，从声明废弃，到彻底删除是一个漫长的过程。在 JDK 中，还存在着大量废弃了 20 多年都无法删除的公开接口。

为什么删除废弃的公开接口这么困难呢？如果从废弃机制本身的角度来思考，下面几个问题延迟了废弃接口使用者的迁移意愿和努力。

第一个问题，也是最重要的问题，就是 SuppressWarnings 注解的使用。

SuppressWarnings 注解的本意是消除编译警告，保持向后的编译兼容性。可是一旦编译警告消除，SuppressWarnings 注解也就抵消了 Deprecated 注解的功效。代码的维护者一旦使用了 SuppressWarnings 注解，就很难再有更合适的工具，让自己知道还在使用的废弃接口有哪些了。不知道，当然就不会有行动。

第二个问题，就是废弃接口的使用者并不担心使用废弃接口。虽然我们都知道不应该使用废弃的接口，但是因为一些人认为没有紧急迁移的必要性，也不急着制定代码迁移的时间表，所以倾向于先使用 SuppressWarnings 注解把编译警告消除了，以后再说迁移的事情。然后，就掉入了第一个问题的陷阱。

第三个问题，就是废弃接口的使用者并不知道接口废弃了多久。在接口使用者的眼里，废弃了十年，和废弃了一年的接口，没有什么区别。可是，在接口维护者的眼里，废弃了十年的接口，应该可以放心地删除了。然而，使用者并没有感知到这样的区别。没有感知，当然也就没有急迫感了。

一旦一个接口被声明为废弃，它的问题也就再难进入接口维护者的任务列表里了。所以，这个接口的实现可能充满了风险和错误。于是局面就变成了，接口维护者难以删除废弃的接口，接口的使用者又不能获得必要的提示，这种情况实在有点尴尬。

改进的废弃

上面这些问题，在 JDK 9 的接口废弃机制里有了重大的改进。

第一个改进是添加了一个新的工具，jdeprscan。有了这个工具，就可以扫描编译好的 Java 类或者包，看看有没有使用废弃的接口了。即使代码使用了 SuppressWarnings 注解，jdeprscan 的结果也不受影响。这个工具解决了我们在阅读案例里提到的第一个问题。

另外，如果我们使用第三方的类库，或者已经编译好的类库，发现对废弃接口的依赖关系很重要。如果将来废弃接口被删除，使用废弃接口的类库将不能正常运行。而 jdeprscan 允许我们在使用一个类库之前进行废弃依赖关系检查，提前做好风险的评估。

第二个改进是给 Deprecated 注解增加了一个 “forRemoval” 的属性。如果这个属性设置为 “true”，那就表示这个废弃接口的删除已经提上日程了。两到三个版本之后，这个废弃的接口就会被删除。这样的改进，强调了代码迁移的紧急性，它给了使用者一个明确的提示。这个改进，解决了我们在阅读案例里提到的第二个问题。

第三个改进是给 Deprecated 注解增加了一个 “since” 的属性。这个属性会说明这个接口是在哪一个版本废弃的。如果我们发现一个接口已经废弃了三年以上，就要考虑尽最大努力进行代码迁移了。这样的改进，给了废弃接口的使用者一个时间上的概念，也方便开发者安排代码迁移的时间表。这个改进，解决了我们在阅读案例里提到的第三个问题。

下面的这段代码，就是一个使用了这两种属性的例子。

 复制代码

```
1 public sealed abstract class Digest {
2     /**
3      * -- snipped
4      *
5      * @deprecated This method is not performance friendly. Use
6      *              {@link #digest(byte[], byte[])} instead.
7      */
8     @Deprecated(since = "1.4", forRemoval = true)
9     public abstract byte[] digest(byte[] message);
10
11     // snipped
12     public void digest(byte[] message, byte[] digestValue) {
13         // snipped
14     }
15 }
```

如果在 `Deprecated` 注解里新加入 “`forRemoval`” 属性，并且设置为 “`true`”，那么以前的 `SuppressWarnings` 就会失去效果。要想消除掉编译警告，我们需要使用新的选项。就像下面的例子这样。

[复制代码](#)

```
1 @SuppressWarnings("removal")
2 public static void main(String[] args) {
3     try {
4         Digest.of("SHA-256")
5             .digest("Hello, world!".getBytes());
6     } catch (NoSuchAlgorithmException ex) {
7         // ignore
8     }
9 }
```

当一个废弃接口的删除提上日程的时候，添加 “`forRemoval`” 属性让我们又有一次机会在代码编译的时候，重新审视还在使用的废弃接口了。

废弃三部曲

有了 JDK 9 的废弃改进，我们就能够看到接口废弃的一般过程了。

第一步，废弃一个接口，标明废弃的版本号，并且描述替代方案；

第二步，添加 “`forRemoval`” 属性，把删除的计划提上日程；

第三步，删除废弃的接口。

对于接口的使用者，我们应该尽量在第一步就做好代码的迁移；如果我们不能在第一步完成迁移，当看到第二步的信号时，我们也要把代码迁移的工作提高优先级，以免影响后续的版本升级。

对于接口的维护者，我们需要尽量按照这个过程退役一个接口，给接口的使用者充分的时间和信息，让他们能够完成代码的迁移。

总结

好，到这里，我来做个小结。刚才，我们讲了接口废弃的现实问题，以及接口废弃的三部曲。总体来说，我们要管理好废弃的接口。接口的废弃要遵守程序，有序推进；代码的迁移要做好计划，尽快完成。


另外，我们要使用好 `jdeprscan` 这个新的工具。在使用一个类库之前，要有意识地进行废弃依赖关系检查，提前做好代码风险的评估。

如果面试中聊到了接口废弃的问题，你可以聊一聊接口废弃的三部曲，以及每一步应该使用的 Java 注解形式。

思考题

今天的思考题，我们来练习一下接口废弃的过程。前面，我们练习过表示形状的封闭类。假设要废弃表示正方形的许可类，我们该怎么做呢？代码该怎么改动呢？

为了方便你阅读，我把表示形状的封闭类的代码拷贝到了下面。请再一次阅读“废弃三部曲”这一小节，然后试着修改下面的代码。

 复制代码

```
1 package co.ivj.jus.retire.review.xuelel;
2
3 public abstract sealed class Shape {
4     public final String id;
5
6     public Shape(String id) {
7         this.id = id;
8     }
9
10    public abstract double area();
11
12    public static final class Circle extends Shape {
13        public final double radius;
14
15        public Circle(String id, double radius) {
16            super(id);
17            this.radius = radius;
18        }
19
20        @Override
21        public double area() {
22            return Math.PI * radius * radius;
23        }
24    }
```



```
25     public static final class Square extends Shape {
26         public final double side;
27
28         public Square(String id, double side) {
29             super(id);
30             this.side = side;
31         }
32
33         @Override
34         public double area() {
35             return side * side;
36         }
37     }
38
39     // Here is your code for Rectangle.
40
41     // Here is the test for circle.
42     public static boolean isCircle(Shape shape) {
43         // Here goes your update.
44         return (shape instanceof Circle);
45     }
46
47     // Here is the code to run your test.
48     public static void main(String[] args) {
49         // Here is your code.
50     }
51 }
52 }
```

欢迎你在留言区留言、讨论，分享你的阅读体验以及你的设计和代码。我们下节课见！

注：本文使用的完整的代码可以从 [🔗 GitHub](#) 下载，你可以通过修改 [🔗 GitHub](#) 上 [🔗 review template](#) 代码，完成这次的思考题。如果你想要分享你的修改或者想听听评审的意见，请提交一个 GitHub 的拉取请求（Pull Request），并把拉取请求的地址贴到留言里。这一小节的拉取请求代码，请在 [🔗 接口废弃专用的代码评审目录](#) 下，建一个以你的名字命名的子目录，代码放到你专有的子目录里。比如，我的代码，就放在 retire/review/xuelel 的目录下面。

分享给需要的人，Ta 订阅后你可得 **20 元现金奖励**

 生成海报并分享

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 15 | 现代密码：你用的加密算法过时了吗？

下一篇 用户故事 | 保持好奇心，积极拥抱变化

精选留言 (2)

💬 写留言

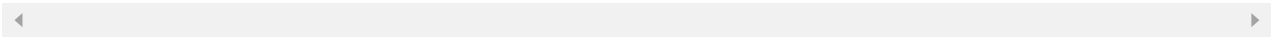


aoe

2021-12-22

大部分项目自身的Bug数远远超过废弃接口可能带来的Bug，所以很难及时清理。

作者回复：没太了解这个逻辑



共 2 条评论 >

👍 1

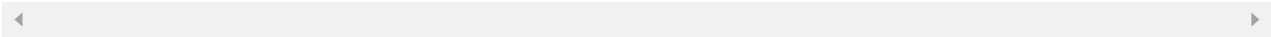


xiaobang

2021-12-22

非jdk库的接口也可以使用这种废弃机制吗？

作者回复：是的



共 2 条评论 >

👍