

## 09 | 怎么用好Java注解？

2019-01-23 范学雷



讲述：刘飞

时长 08:03 大小 18.45M



如果你使用面向对象的概念和技术有一段时间了，不知道你会不会有这样的困惑：面向对象技术带来的麻烦，一点都不比它带来的好处少！

比如说，我们辛辛苦苦继承了一个类，重写了它的方法。过几天，这个类居然修改了它的接口，而且没人通知我们。然后，我们写的子类还可以美滋滋地编译，运行，就是总出错误，怎么调试都没发现这个子类的实现有什么不妥。直到有人说，父类变了！这时候，我们就想找杯咖啡暖暖手，一个人静静。

面向对象技术确实有它值得傲娇的地方。但是，只有把类似上面的小麻烦解决掉，它的使用才更合理。比如说，父类做了修改，能不能立即就通知我？别等到问题出现了，我们还被蒙在鼓里。

Java 注解就可以帮助我们。

# 什么是 Java 注解

Java 注解是 Java 1.5 引入的一个工具，类似于给代码贴个标签，通过注解可以为代码添加标签信息。这些标签信息可以添加在字段、方法和类上。开发工具、部署工具或者运行类库，可以对这些标签信息进行特殊的处理，从而获得更丰富的功能。


经过十多年的发展，注解已经成了 Java 生态系统一个非常重要的技术。使用注解可以大幅度降低我们的开发强度，提高工作效率，减少潜在的错误。像 Java 类库一样，注解也有了越来越丰富的定义和规范，成了我们需要掌握的重要技术之一。

**我们这里只讨论编写规范的代码时，该怎么合理地使用注解，具体就是 `Override`、`Deprecated`、`SuppressWarnings` 这三个注解。**更详细的 Java 注解技术和规范，以及如何自定义注解，需要你参考相关的文档。

## 在声明继承关系中，Java 注解该如何使用？

在代码编写中，继承和重写是面向对象编程的两个重要的机制。这两个机制，在给我们带来便利的同时，也顺便带来了一些麻烦，这就需要我们用到注解了。

**第一个麻烦是，识别子类的方法是不是重写方法。**比如下面的例子，在一般情况下，对代码阅读者来说，最直觉的感受就是，`getFirstName()` 这个方法不是重写方法，父类 `Person` 没有定义这个方法。

 复制代码

```
1 class Student extends Person {
2     // snipped
3     public String getFirstName() {
4         // snipped
5     }
6     // snipped
7 }
```

通常如果一个方法是重写方法，一定要使用 `Override` 注解，清楚地标明这个方法是重写的方法。使用 `Override` 注解的另一个好处是，如果父类更改了方法，子类的编译就会出错。这样我们就能在第一时间获得通知，既可以及时地变更子类，也可以使父类的变更更加合理。

```
1 class Student extends Person {
2     // snipped
3     @Override
4     public String getFirstName() {
5         // snipped
6     }
7     // snipped
8 }
```

为什么要识别重写方法呢？这是因为继承的第二个麻烦。

**第二个麻烦是，重写方法可以不遵守父类方法的规范。**面向对象编程的机制，理想的状况是，父类定义了方法和规范，子类严格地遵守父类的定义。比如 `Person.getFirstName()` 要求返回值是一个人的名，不包括姓氏部分，而且不可以是空值。但是子类 `Student.getFirstName()` 的实现完全有可能没有严格遵守这样的规范，不管是有意的，或者是无意的。比如，返回了姓氏，或者返回了包括姓氏的姓名，或者可以返回了空值。

```
1 class Student extends Person {
2     // snipped
3     @Override
4     public String getFirstName() {
5         return null;
6     }
7     // snipped
8 }
```

编译器无法检查重写到底该怎么实现，保持重写方法的行为一致需要我们凭借经验、肉眼识别。一般来说，一个重写方法不应该改变父类定义的规范。如果的确需要改变，就要有充足的理由，以及面对潜在兼容问题的具体的解决办法。

比如上面的例子中，如果 `Person.getFirstName()` 不允许返回空值，应用程序可以很安心地使用返回值，而不需要检查空值。

```
1 boolean isAlice(Person person) {
2     return person.getFirstName().equals("Alice");
3 }
```

但是，有了可以返回空值的 `Studen.getFirstName()` 的重写，上面的代码就可能抛出 `NullPointerException`。一段简单的、严格遵守规范的代码，就变得危机四伏。

既然需要肉眼的判断，第一步就是要识别出重写方法。识别方法越简单越好。

所以，重写的方法，一定要加上 `Override` 注解。这个注解，既可以提醒代码的读者，也提醒代码的书写者，要谨慎对待该方法在父类定义规范。

识别出重写方法后，第二步就要判断重写的方法和父类规范的定义有没有冲突和抵触。

虽然一般情况下，子类的重写方法不应该改变父类的规范。但是，编写代码处处充满了无奈和妥协。极少数情况下，除了变更方法的规范，我们可能别无选择。一旦这种情况发生，一定要明确标明，并注明潜在的后果。

如果重写方法既没有改变父类规范，也没有其他情况需要重点说明，重写方法就不应该有规范描述部分的存在。这样，可以减少规范描述对于读者的误导。我们当然需要了解具体的规范，但是应该查找、阅读父类的规范描述。

反面案例	<pre>class Student extends Person {     public String getFirstName() {         return null;     } }</pre>	没有使用 Override 注解
反面案例	<pre>class Student extends Person {     /**      * Get the first name of the student.      */     @Override     public String getFirstName() {         // snipped     } }</pre>	多余的规范描述，干扰阅读者的判断
反面案例	<pre>class Student extends Person {     @Override     public String getFirstName() {         return null;     } }</pre>	没有表明重写方法规范的变更
正面案例	<pre>class Student extends Person {     @Override     public String getFirstName() {         // snipped     } }</pre>	重写方法严格遵守父类定义的规范
正面案例	<pre>private class Student extends Person {     /**      * Get the first name of the student.      *      * Note that the returned value may      * be null if ...      */     @Override     public String getFirstName() {         // snipped         return null;     } }</pre>	重写方法改写了父类定义的规范

继承和重写还有一些其他的麻烦，我们后面的章节接着再聊。


## 在废弃退役接口的情况下，如何使用注解？



一个软件，部署得越广泛，生命力越悠久，就越需要不断地改进、升级。而废弃不合理的设计，拥抱更新的思想，也是软件改进的一部分。

然而，软件接口的废弃，不是一件简单的事情。越是广泛使用的接口，它的废弃、退役越困难。

比如，下面的 String 构造方法，是 1994 年 Java 1.0 设计实现的方法。很快，人们发现了这个方法的局限性。在 1997 年发布的 Java 1.1 中，废弃了该构造方法以及其他相关的方法。到现在，已经废弃 20 多年了，但 Java 依然没有删除这些方法，因为 String 的使用太广泛了！

 复制代码

```
1 @Deprecated(since="1.1")
2 public String(byte ascii[], int hibyte) {
3     this(ascii, hibyte, 0, ascii.length);
4 }
```

无论对于软件的维护者，还是软件的使用者，废弃的接口都是不值得让我们继续耗费精力的。

如果软件的维护者继续在废弃的接口上投入精力，意味着这个接口随着时间的推移，它的实现可能会存在各种各样的问题，包括严重的安全问题，就连使用者也要承担这些风险。而且还会有用户持续把它们运用到新的应用中去，这就违背了废弃接口的初衷。更多的使用者加入危险的游戏，也增加了删除废弃接口的难度。


这就要求我们做好两件事情。

**第一件事情是，如果接口的设计存在不合理性，或者新方法取代了旧方法，我们应该尽早地废弃该接口。**


及时止损！

做好这件事情，需要我们使用 Deprecated 注解，并且用一切可以使用的办法，广而告之。对于代码而言，要在声明中使用 Deprecated 注解；在规范描述中，说明废弃的原因以及替代的办法；对于有计划要删除的接口，要注明计划删除的版本号。

下面是两个可以参照的 Java 代码废弃接口的例子：

 复制代码

```
1 java/lang/String.java:
2
3 /**
4  * Counts the number of stack frames in this thread. The thread must
5  * be suspended.
6  *
7  * @return    the number of stack frames in this thread.
8  * @throws    InterruptedException if this thread is not
9  *           suspended.
10 * @deprecated The definition of this call depends on
11 *             {@link #suspend}, which is deprecated. Further,
12 *             the results of this call were never well-defined.
13 *             This method is subject to removal in a future
14 *             version of Java SE.
15 * @see       StackWalker
16 */
17 @Deprecated(since="1.2", forRemoval=true)
18 public native int countStackFrames();
```


 复制代码

```
1 java.security.Certificate.java:
2
3 /**
4  * <p>This is an interface of abstract methods for managing a
5  * variety of identity certificates.
6  *
7  * ... snipped ...
8  *
9  * @deprecated This class is deprecated and subject to removal
10 *             in a future version of Java SE. It has been replaced by
11 *             {@code java.security.cert.Certificate} and related classes.
12 * @see java.security.cert.Certificate
13 */
14 @Deprecated(since="1.2", forRemoval=true)
15 public interface Certificate {
16     // snipped
17 }
```

**第二件事情是，如果我们在现有的代码中使用了废弃的接口，要尽快转换、使用替换的方法。等到废弃方法删除的时候，再去更改，就太晚了，不要等到压力山大的时候才救火。**

如果一个接口被废弃，编译器会警告继续使用的代码。Java 提供了一个不推荐使用的注解，`SuppressWarnings`。这个注解告诉编译器，忽略特定的警告。警告是非常有价值的信息，忽略警告永远不是一个最好的选项。

再次强调，除非万不得已，不要使用 `SuppressWarnings`。如果万不得已来临，请参考下面的例子。

 复制代码

```
1 @SuppressWarnings("deprecation")
2 private boolean myMethodUseDeprecatedMethod() {
3     // snipped
4 }
```

当然，这样的使用带来了一系列的后遗症。由于，废弃的编译警告被无视，我们使用了废弃接口的事实就被淹没在代码的海洋里，再也进入不了我们的视野。不到废弃接口被删除的那一天，我们都意识不到我们的代码里使用了废弃的接口，我们的应用程序都要承担着废弃接口维护不足的种种风险，包括严重的安全风险。

后面我们还会谈到，不要轻易地更改现有的代码，即使这些代码很丑陋，散发着浓浓的腐臭味。但是，有一个例外，如果看到了使用 `SuppressWarnings` 的代码，我们要尽可能地想办法把相关的警告消除掉、把这个注解去掉，越快越好。

## 小结

Java 注解的功能很丰富，了解注解可以使得我们编码的工作更轻松。这一次，希望我们记住三个基本的实践：

1. 重写的方法，总是使用；
2. 过时的接口，尽早废弃；
3. 废弃的接口，不要使用。

## 一起来动手

Java 的注解非常丰富，功能也很强大。借这个机会，我想让大家互相分享一下，你最经常使用的注解是什么？什么情况下使用这个注解？这个注解给你带来哪些便利？欢迎你把你的经验发布在评论区，我们一起来学习更多的注解，一起来进步。



也欢迎你把这篇文章分享给你的朋友或者同事，一起来探讨吧！



# 代码精进之路

你写的每一行代码都是你的名片

范学雷

Oracle 首席软件工程师  
Java SE 安全组成员  
OpenJDK 评审成员



新版升级：点击「👤 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得转载

上一篇 08 | 写好声明的“八项纪律”

下一篇 10 | 异常处理都有哪些陷阱？

## 精选留言 (16)

💬 写留言



hua168

2019-01-23

👍 12

老师，问3个很重要的题外问题：

1. 大专学历，想直接自学考本科或研究生，自考学历中大型公司承认的吗？
  2. 大公司对年龄有限制的吗？
  3. 30多岁，运维（编程自学java一年，没项目经验），只有小公司工作经验，技术一般，发展方向是什么？很多IT公司好像都不要年龄大点的~~人生80，那不是40岁就没得工...
- 展开 ∨

作者回复: 人生三十，学历的分量应该比重不大了。国内的部分大公司，年龄是个考量的因素，但是也就是众多因素中的一个因素。如果把劳动力看成一个理性的市场，重要的是我们有什么可

以拿来交换的。

我有一个朋友，认识他的人都特别佩服，他曾经兼顾过一段时间的非常枯燥无聊、看起来也没什么技术含量的工作。过了几年，当他把工作转交给下一个人时，他已经把这项工作做的像一个艺术了。他做了大量的自动化，看起来非常枯燥无聊的事情，被他搞的很有趣，自动化后，根本就占用不了他多长时间。

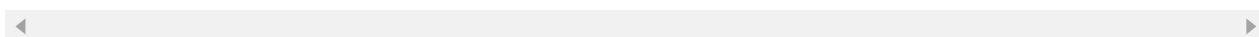
软件是一个复杂的系统，需要各种各样的人才，有能研发的，有能测试的，有能运维的，有能运营的。每种都需要不同的技能。再厉害的研发也不一定做得了测试，也不一定做得了运维。

把你手头的工作做出花来，这就是我们值钱的地方。怎么才能做出花来呢？这个你的领导不知道，你的公司可能也不知道，所以大概率没有人能够教你怎么做。你得自己去从工作中发现现实的问题，找到具体的解决办法。多看看新的技术，想一想能不能用到你的工作环境中去。你可能看到100个技术，只有一个能用。一个能用，这就行了。你要是不看100个，可能没有办法找到这一个。

运维就是一个涉及到很多复杂知识的岗位，它的复杂度一点都不比软件开发简单。但是，的确也有的运维就是看几台机器。但是，既然是你做运维，你可以自己定义运维该怎么做最好。

不管时哪一个岗位，如果做到一定程度，年龄从来都不是真正的问题。很多岗位，年龄都是优势，而不是问题，因为没有积年累月的积累，我们成不了一个领域最专业的人。成为问题的，是我们40岁了，水平还停留在20到30岁的阶段。

Stay Hungry, Stay Foolish!



岁月安然

2019-01-24

7

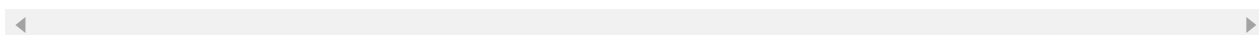
lombok插件的很多实用的注解

@Data 使用在类上，该注解会提供getter、setter、equals、canEqual、hashCode、toString方法。

@NonNull 该注解使用在属性上，该注解用于属的非空检查，当放在setter方法的字段上，将生成一个空检查，如果为空，则抛出NullPointerException。...

展开 v

作者回复: 学习了，我还不知道这个插件。注解真的非常强大！





web

2019-01-28

👍 1

题目有点大, 以为是讲怎么写注解; 内容有点水, 半个版面怎么写override

展开 ▾

作者回复: ☺和规范有关的注解。



Being

2019-01-23

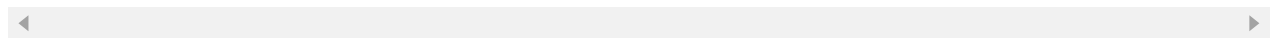
👍 1

Override对应C++就是virtual了, 经常用, 以前还真不太清楚为什么重写的方法要加, 就觉得好区别, 就保留这个习惯了, 今天才意识到要避免父类删除继承方法后, 能快速通过编译器定位问题。

貌似C++没有JAVA的Depraceted和SuppressWarnings类似的, 我再查查确认下, Deprecated的用处挺大的, 及时止损呀

展开 ▾

作者回复: 谢谢分享C++对应的关键字。



王智

2019-01-23

👍 1

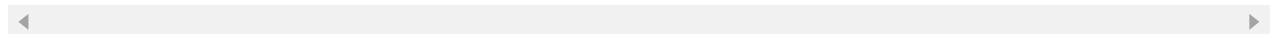
阿里巴巴的扫描插件或许会有帮助的, 在使用idea的过程中, 安装了Alibaba Java Coding Guidelines之后, 代码的规范等等插件就会检测出来报红, 虽然可以运行, 但是对于有强迫症和代码洁癖的人来说就很难受。

包括了if不写括号, 继承的方法没有使用@Override注解。

...

展开 ▾

作者回复: 加油!



fighting

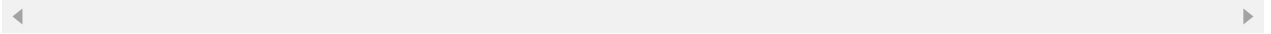
2019-01-23

👍 1

课程都是以java讲解的吗, 没有其他编程语言吗

展开 ▾

作者回复: 除了个别的例子，几乎都是用Java讲解的。



苦行僧

2019-01-23

👍 1

现在基本上是用静态代码检查工具扫描业务代码，jdk中的废弃方法基本替换掉  
展开 ▾

作者回复: 赞，这是一个好习惯！



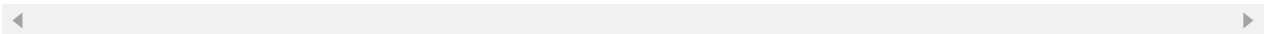
悲劇の輪廻

2019-02-25

👍

虽然只是为了举例而写的代码段，但一般情况下需要被equals的对象为字符串时我们会声明一个字符串常量，而当一个从方法获得的值需要与字符串常量作比较的时候，通常把方法返回值放在被equals的位置，这样也能避免方法返回null时抛出空指针异常，而且不会引起逻辑上的错误。:-)

作者回复: 只有多次使用时，一般才会考虑定义常量。equals的使用是一个很好的实践。代码质量就是这些一点一点的小事累积而成的。



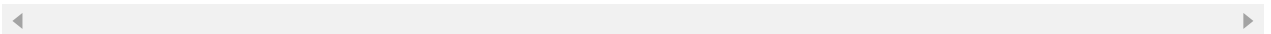
醉侠

2019-02-13

👍

老师，想知道文章里String的构造函数为什么被移除，是因为字符编码的问题吗？  
展开 ▾

作者回复: 是字符编码的问题，这个构造函数（String(byte[] ascii, int hibyte)）没有充分的信息可以把字节（byte）转换成合适的字符（character）。



小新是也

2019-02-03

👍

用的最多的应该是Lombox的@Data了

展开 ∨



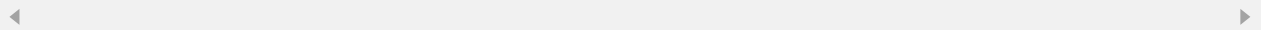
小成

2019-01-30



C++11引入了override关键字，对应Java的override.  
C++14引入了deprecated关键字，对应Java的deprecated.

作者回复: 谢谢分享，这个留言对使用C++的有帮助！



hua168

2019-01-24



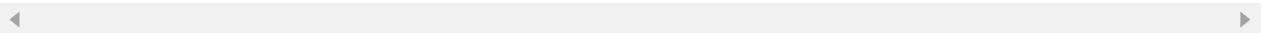
看你介绍，您是DBA大神，有哥们小公司搞DBA几年，会mysql、MSSQL、mongoDB，群集、分库，分表、分区简单优化等，不懂开发，我想问一下：

- 1.DBA一般发展方向是怎样的呀？运维和开发我了解，DBA没接触过，无法给建议，一般的升级过程是怎样的？
- 2.DBA开发语言选择是C++还是java，还是其它？...

展开 ∨

作者回复: 可能是你搞错了，我不懂Oracle的数据库。我的数据库知识也比较陈旧了，还是十多年前的见识，没什么长进。

小伙伴们帮着回答一下吧！



hua168

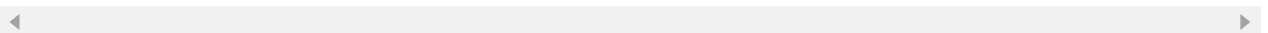
2019-01-24



非常感谢您的认真回答！谢谢.....

展开 ∨

作者回复: 不客气



多拉格 fj4

在项目中使用Lombok可以减少很多重复代码的书写。使用注解在class文件中生成getter/setter/toString等方法。



作者回复: 了解了。谢谢！



www

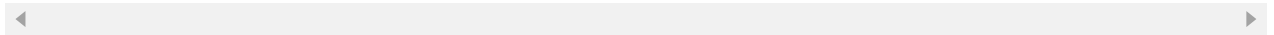
2019-01-23



lombok

展开 ▾

作者回复: lombok是什么？怎么用？能分享多一点内容吗？



allean

2019-01-23



Findbugs？

展开 ▾

作者回复: FindBugs/SpotBugs，或者规范的插件，都能检查出来这些问题。

