> 本文由 简悦 SimpRead 转码， 原文地址 www.imooc.com

《手册》第 7 页对于过时类有这样一句描述：

> 接口过时必须加 @Deprecated 注解，并清晰地说明采用的新接口或者新服务是什么。
> 接口提供方既然明确是过时接口，那么有义务同时提供新的接口；作为调用方来说，有义务去考证过时方法的新实现是什么。

那么我们要思考为什么要这么做呢？这个指导原则如何更好地落地呢？

如果有机会进入一个大一点的公司，而且你是一个有追求的人，你可能会遇到下面几种情况。

- 当你接手一个服务，看到某个类、属性、函数被标注为 @Deprecated 但是没有注释的时候，内心是崩溃的；
- 当你对接二方服务，升级 jar 包后发现使用的接口被标记为废弃但是没注释时，内心也是崩溃的；
- 当你看到同事封装的一些工具类使用了一些被废弃的类时，你的内心同样同样是崩溃的。不改放在那看着难受，改又无故得耗费自己的时间，而且还怕改出 BUG。

试想一下，如果你接手一个服务里面的类、属性和函数要被废弃了连 @Deprecated 都不加，是不是很容易 “放心” 使用进而被坑？

如果被标注为 `@Deprecated` ，给出注释说明为什么被废弃，新的接口是什么，心里会不会更踏实？

如果对接的二方服务 jar 包升级以后发现，使用的接口被废弃且给出详细的告诉你改用哪个新接口，是不是心里更有底？

试想一下如果我们每个人都能遵守这种规约，封装工具类时遇到过时的类，主动去学习并使用新的替换类，是不是就不会好很多？

那么，说了这么多，究竟该如何落地呢？
我认为：最好的学习方式之一就是找一些优秀的源码相关的示例进行学习。

3.1 JDK 的类或常见三方库

我们以 JDK 中的 `URLEncoder` 和 `URLDecoder` 为例介绍如何写过期函数的注释和如何替换该过期函数：

```
String url = "xxx";
String encode = URLEncoder.encode(url);
log.debug("URL编码结果：" + encode);
String decode = URLDecoder.decode(encode);
log.debug("URL解码结果：" + decode);
```

在 IDEA 中编写如上代码时候，`java.net.URLEncoder#encode(java.lang.String)` 和 `java.net.URLDecoder#decode(java.lang.String)` 会有删除的标志，便表示该函数已经过期。

那么如何找到新函数和修改呢？

我们进到源码里查看：

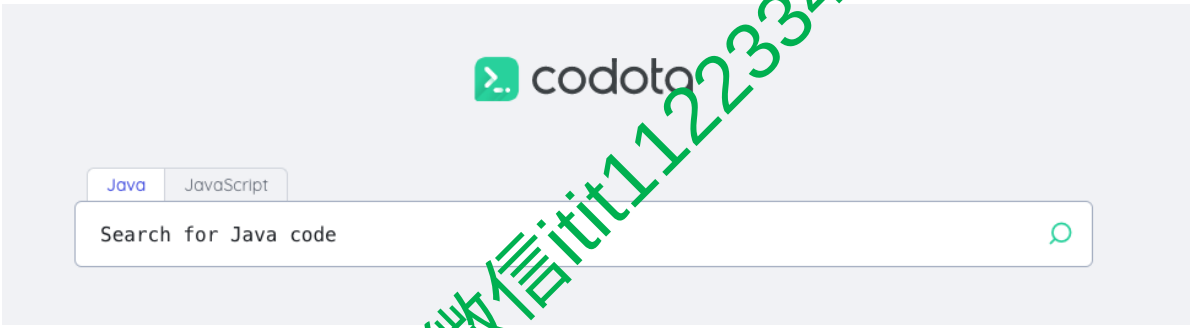
```
@Deprecated
public static String decode(String s) {
    String str = null;
    try {
        str = decode(s, dfltEncName);
    } catch (UnsupportedEncodingException e) {

    }
    return str;
}
```

在 `@deprecated` 的注释里我们找到了答案：“The resulting string may vary depending on the platform’s default encoding.（解析结果的字符串和系统的默认字符编码强关联）”，并给出了替代函数的说明 “Instead, use the `decode(String,String)` method to specify the encoding.（使用 `decode(String,String)` 函数来指定字符串编码）”

因此我们提供新的接口，就得接口要废弃时也可以参考这里**写上废弃的原因以及替代的新接口。**

我们还可以通过 codota 来搜索（建议在 IDEA 安装插件，使用更方便）看常见类库的常见函数的用法，甚至可以看到某些函数的使用概率：



搜索我们想要的类和方法：URLEncoder.encode，即可得到 github 优秀的开源框架或 stackoverflow 中相关优秀范例。根据相关的优秀代码范例进行修改。

我们改用新的函数：

```
    String url = "xxx";
    String encode = URLEncoder.encode(url, Charsets.UTF_8.name());
    log.debug("URL编码结果：" + encode);
    String decode = URLDecoder.decode(encode, Charsets.UTF_8.name());
    log.debug("URL解码结果：" + decode);
```


对类似废弃的接口的改动，最好要使用单元测试进行验证：

```
@Test
public void testURLUtil() throws UnsupportedEncodingException {

    String url = "http://www.imooc.com/test?;

    String encodeOrigin = URLEncoder.encode(url);
    String decodeOrigin = URLDecoder.decode(encodeOrigin);


    String encodeNew = URLEncoder.encode(url, Charsets.UTF_8.name());
    String decodeNew = URLDecoder.decode(encodeNew, Charsets.UTF_8.name());


    Assert.assertEquals(encodeOrigin, encodeNew);
    Assert.assertEquals(decodeOrigin, decodeNew);
}
```

如果是常见的**三方库**，也可以采用类似的步骤，一般都很快解决问题。

如我们发现下面的函数被废弃，进入到源码中查看：

`org.springframework.util.Assert#doesNotContain(java.lang.String, java.lang.String)`

```
@Deprecated
public static void doesNotContain(String textToSearch, String substring) {
   doesNotContain(textToSearch, substring,
         "[Assertion failed] - this String argument must not contain the
substring [" + substring + "]");
}
```

直接通过点击 `{@link #doesNotContain(String, String, String)` 可以快速进入新的替代函数去查看。

从这里例子我们还学到了一个新的技巧，如果是二方库或者三方库，废弃的属性、函数在注释中除了可以写原因和替代函数外，可以标注从哪个版本被标注为废弃。替代函数可以使用 `{@link}` 方式，更便捷和优雅。

再回顾上面 `java.net.URLDecoder#decode(java.lang.String)` 的注释就没有提供这种方式，跳转就不够方便。

另外大家还可以学习一下 `@see` 的用法，以及 `@see` 和 `{@link}` 的区别，后面专栏也会对注释做专门的讲解。

我们从这个例子还可以看到注释中并没有说明废弃的原因，作为读者你会发现有些摸不着头脑，心里嘀咕“为啥被废弃？”。

通过替换函数以及注释我们可以猜测废弃的原因是：”默认的提示文本不够优雅“且即使断言通过，第三个参数字符串拼接仍然会执行，造成不必要字符串连接操作。这点有点类似于日志中不建议使用字符串拼接当做日志内容（可以采用占位符的方式）。

新的替换函数的注释除了给出功能介绍外，也给出了使用的范例：

```
Assert.doesNotContain(name, "rod", "Name must not contain 'rod'");
```

这里给我们带来的启发是，**写工具类时如果能再注释上添加一些范例和结果，则会极大方便使用者。**

这点在 `commons-lang3` 和 `guava` 等开源工具库中随处可见，值得我们学习。

随手选取一个例子，大家感受一下：

```
public static String stripToNull(String str) {
    if (str == null) {
        return null;
    }
    str = strip(str, null);
    return str.isEmpty() ? null : str;
}
```

对于常见的三方库，还有一个不错的技巧：我们可以从 github 上拉取其源代码，然后找到某个类对应的单元测试类中，在单元测试模块可以找到对应的参考用法。还可以在源码中打断点，进行深入研究。希望大家可以亲自实践，会有更加深刻的体会。

3.2 二方库

作为接口的使用者，如果使用二方库，发现使用的功能被标注为废弃。

如果是 maven 项目可以通过 maven 命令拉取其源码和 javadoc。

`mvn dependency:sources -DdownloadSources=true -DdownloadJavadocs=true`

如果是 gradle 项目，也可以使用插件下载源码，查看其将被废弃的原因。

如果没有标注原因并给出替代方案，或给出的注释不够详细，建议直接和二方包的提供者联系，及早替换。

二方库的工具类替换成新的接口也必须要通过单测，并对涉及的功能进行回归。

3.3 自己库

作为接口或对象的提供者，废弃的类、属性、函数加上废弃的原因和替代方案。

如 RPC 订单常见接口的 `OrderCreateParam` 参数类的 JSON 类型参数：`orderItemDetail` 要替换成列表 `orderItemParams` 下面的属性类型进行替换：

```
public class OrderCreateParam {


    @Deprecated
    private String orderItemDetail;

    private List<OrderItemParam> orderItemParams;


}
```


自己类的变动要通过单元测试进行验证：

```
@Test
public void testOriginAndNew() {

    OrderCreateParam orderCreateParamOrigin = new OrderCreateParam();

    orderCreateParamOrigin.setOrderItemDetail("[{\"count\":22,\"name\":\"商品
1\"},{\"count\":33,\"name\":\"商品2\"}]");

    OrderCreateParam orderCreateParamNew = new OrderCreateParam();

    List<OrderItemParam> orderItemParamList = new ArrayList<>(2);
    OrderItemParam orderItemParam = new OrderItemParam();
    orderItemParam.setName("商品1");
    orderItemParam.setCount(22);
    orderItemParamList.add(orderItemParam);

    OrderItemParam orderItemParam2 = new OrderItemParam();
```

```
    orderItemParam2.setName("商品2");
    orderItemParam2.setCount(33);
    orderItemParamList.add(orderItemParam2);
    orderCreateParamNew.setOrderItemParams(orderItemParamList);


Assert.assertEquals(JSON.toJSONString(orderCreateParamNew.getOrderItemParams()),
orderCreateParamOrigin.getOrderItemDetail());
}
```

这里给出一个简单的模拟范例，实际业务代码中参数的接口还要进行 mock 单元测试（后续章节会有相关介绍），对应接口要根据变动传入不同的参数进行功能测试。

如如果实际开发中自己需要改动的功能涉及到废弃的类、属性、函数等，且没有详细地注释，无法获知废弃的原因和替代的方案。可以通过 IDEA 的 “annotate” 菜单，或者 “Git” - ”Show History for Selection“ 等来查看添加废弃注解的人员与之联系。避免自己错代码，如果搞明白问题且仍然不能废弃，最好能够主动将废弃的原因和替代的代码补充到注释中。

如果是三方或二方库，由于作者责任性不强或者职业素养不高，对某个接口标记废弃且没有任何注释时，我们优先在本类中寻找函数签名相似的函数。如果是开源项目或者公司内部可以拉取的项目，可以拉取该项目代码，找到该类查看提交记录，从中寻找线索。

不管是三方、二方还是自己的项目，对替换废弃的类、属性和方法等进行修改后，一定要通过单元测试去验证功能并且对接口使用的功能进行功能测试。

如果要删除废弃的属性或接口，一般先提供新的方案通知使用方修改，此时可以在将废弃的接口上加上日志，新旧接口同时运行一段时间后确认无调用再下一个版本中考虑删除接口。

如果我们能快速找到替代的方案，就可以节省很多时间；如果我们能够充分地测试，就可以平稳替换；如果我们能够介绍清楚废弃的原因，提供新的替代方案，并给出快捷的跳转方式，我们的专业程度就会提高。

4. 总结

本节的主要介绍过期类、属性、接口的正确处理姿势，包括添加废弃注解，添加废弃的原因，添加新接口的跳转等方式，还要在替换后对新接口进行测试测试。本小节还介绍了通过查看相关的优秀开源代码、使用 codota 工具来学习相关知识的方法。

下节我们将学习开发中经常碰到的又爱又恨的空指针，了解其产生的主要原因，学习如何尽可能地避免。

从 github 上拉取 Spring 源码，找到 `org.springframework.util.Assert#doesNotContain(java.lang.String, java.lang.String, java.lang.String)` 方法的单元测试代码并运行。

}