1. **注释的作用**
2. **好注释的作用**

弥补我们在用代码表达意图时遭遇的失败，能够提供对代码清晰的表述和说明。

1. **坏注释的作用**

提供错误的提示信息，产生误导。

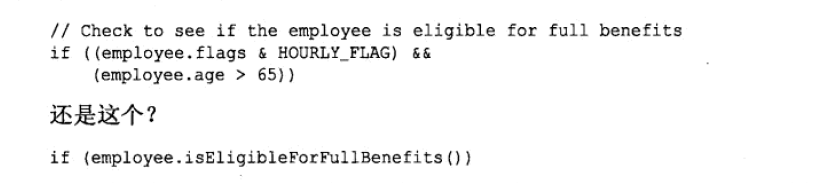
1. **注释的局限**
2. **注释不能美化糟糕的代码**

与其花时间为糟糕的代码添加注释，不如花时间将代码修改干净。

带有少量注释的整洁而有表达力的代码要比带有大量注释的零碎而复杂的代码像样得多。

1. **用代码来阐述**

用描述性/解释性的名称来代替注释。



1. **好注释**
2. **法律信息**

有时候公司代码规范要求编写与法律有关的注释，例如，版权及著作权声明就是必须有理由在每个源文件开头注释处放置的内容。

这类注释不应该是合同或者法典。只要有可能就应该指向一份标准许可或者其他外部文档，而不是把所有条款放在注释中。



1. **提供信息的注释**

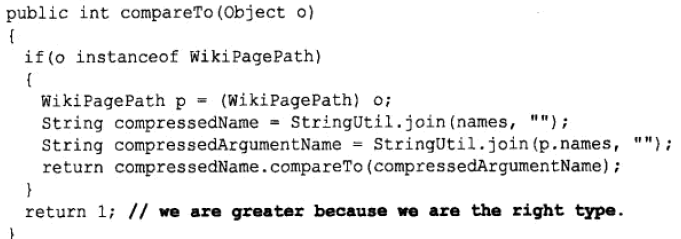
提供一些关于代码的基本信息。例如，以下注释解释了某个抽象方法的返回值。



更好的做法是利用函数名称传达信息，比如本例中只要重新命名函数名为responderBeingTested即可。

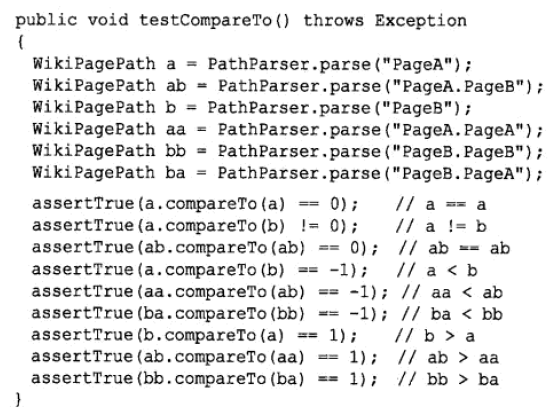
1. **对意图的解释**

注释有时候用来说明某个决定后面的意图。在下例中，在对比两个对象时，作者决定将他的类放置在比其他东西更高的位置。



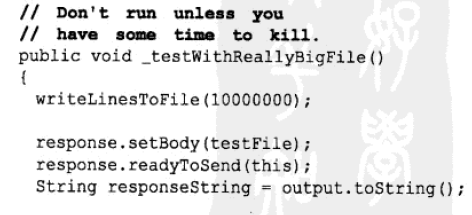
1. **阐释**

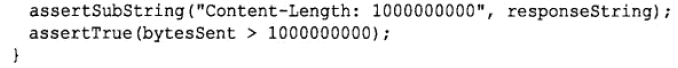
注释将某些晦涩难明的参数或者返回值的意义翻译为某种可读的形式，更好的方法是尽量让参数或者返回值本身就足够清楚。



1. **警示**

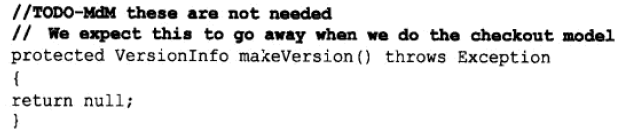
用于警示其他程序员会出现某种后果。





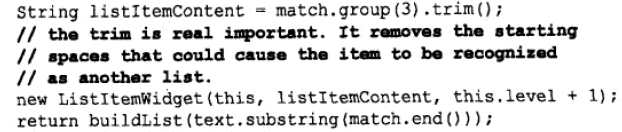
1. **TODO注释**

TODO注释表明程序员应该做，但由于某种原因还没有做的工作。



1. **放大**

注释可以放大某种看似不合理之物的重要性。

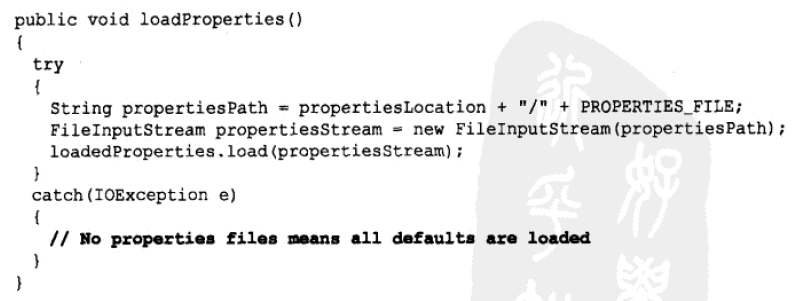


1. **公共API中的javadoc**

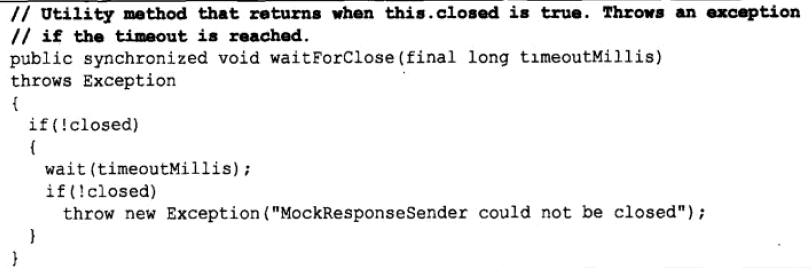
没有什么比良好描述的公共API更有用和令人满意的了。

1. **坏注释**
2. **喃喃自语**

只是因为应该或者因为过程需要就添加注释，那就是无谓之举。



1. **多余的注释**

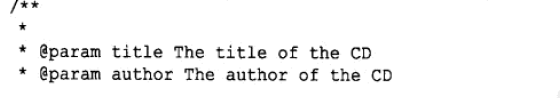


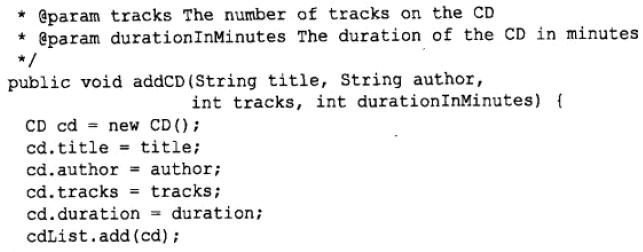
1. **误导性注释**

如上述代码描述，在this.closed变为true时，方法并没有返回。方法只是在判断到this.closed为true时返回。否则就只是等待遥遥无期的超时，然后如果判断this.closed还是非true,就抛出一个异常。

1. **循规式注释**

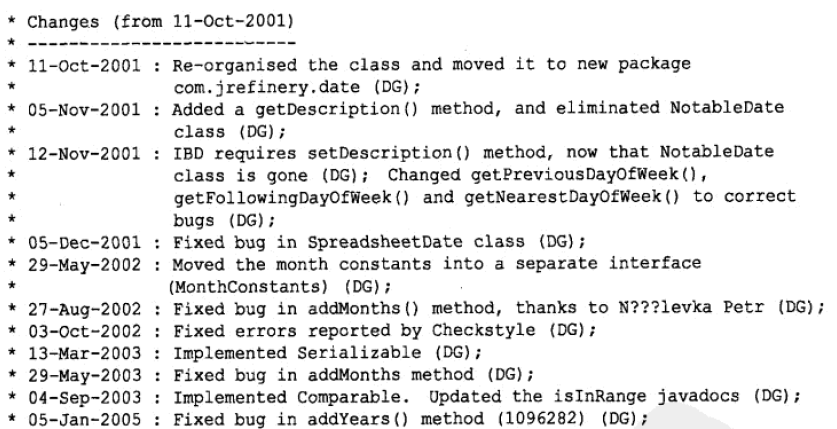
每个函数都要有javadoc或每个变量都要有注释的规矩是愚蠢的。这类注释让代码变得散乱，满口胡言，令人迷惑。





1. **日志式注释**

这类注释就像记录每次修改的日志。

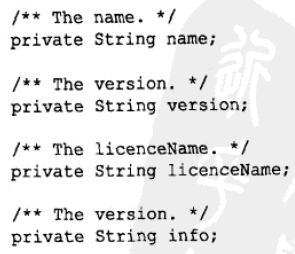


1. **废话注释**

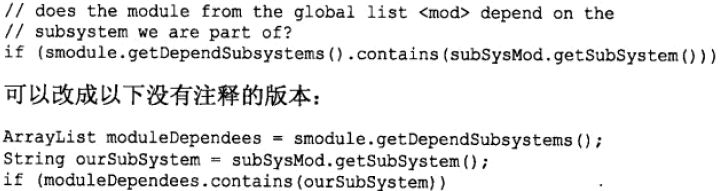
**对于很明显的代码进行解释。**



1. **可怕的废话**



1. **能用函数或者变量时就别用注释**

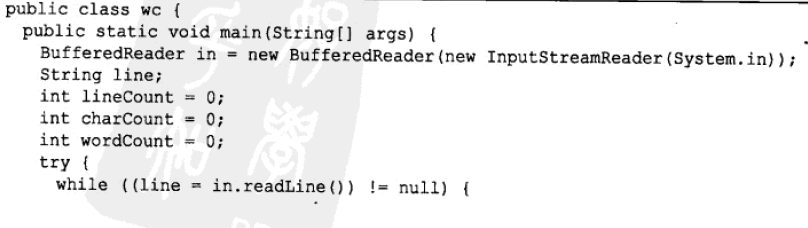


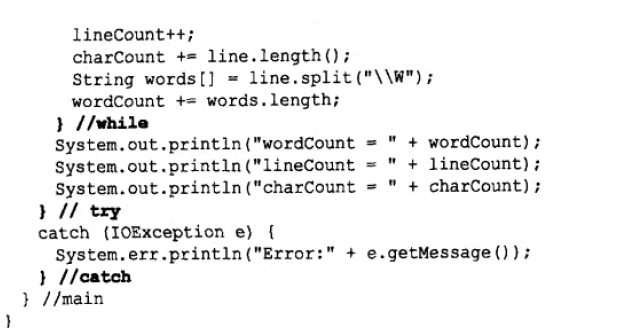
1. **位置标记**

在源代码中标记某个特定的位置。



1. **括号后面的注释**





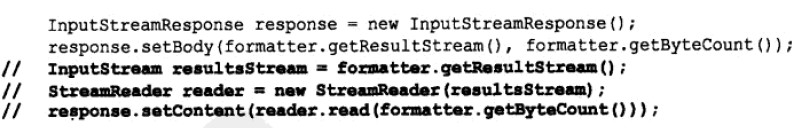
1. **归属与署名**

源代码控制系统非常善于记住是谁在何时添加了什么，没必要用小小的签名来搞脏代码。



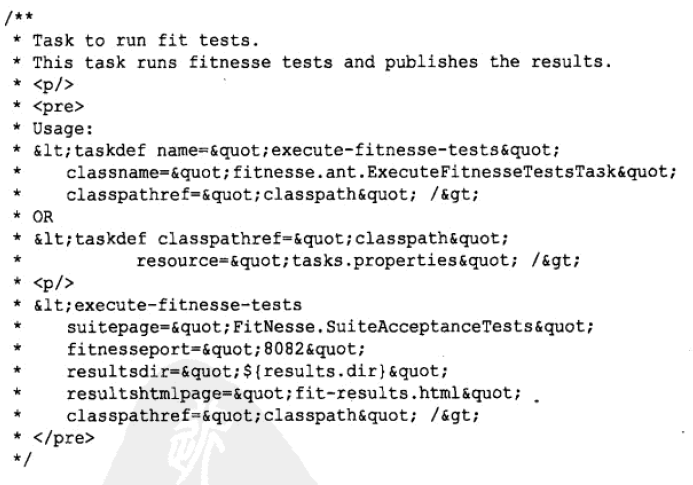
1. **注释掉的代码**

不要直接注释掉代码。注释掉的代码会给其他程序员带来疑惑。为什么代码被注释掉？这些代码重要吗？以后是否会用到？



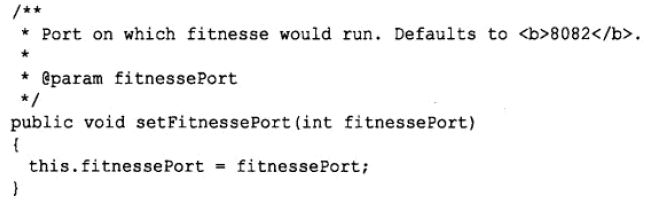
1. **HTML注释**

HTML注释难以阅读。



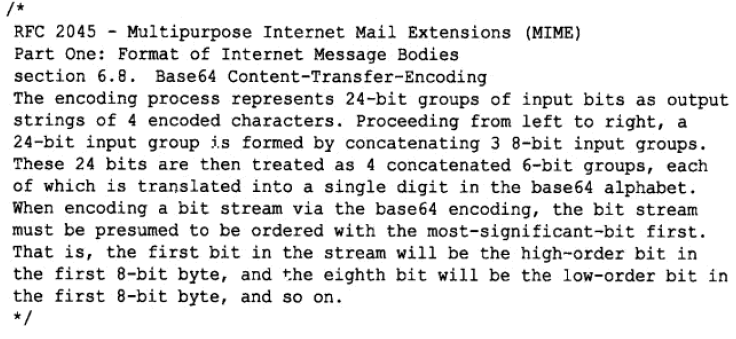
1. **非本地信息**

注释没有在代码实际所在处进行描述。



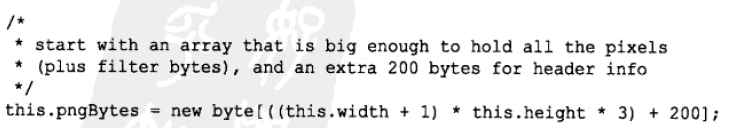
1. **信息过多**

**注释中添加过多的细节描述或者历史描述**



1. **不明显的联系**

注释与所描述的代码之间的关系不够显而易见。



1. **函数头**

为函数头添加注释。较好的做法是创建短小函数并起一个合理的名字。

1. **非公共代码的javadoc**

为系统中非公共代码生成javadoc