什么叫可读性

曾经看到过这样一句话「别人在阅读代码过程中飙脏话的频率是衡量你代码质量的唯一标准」。

代码的可读性其实不是针对的编译器、解释器，而是对于人来说的。具有良好可读性的代码，应该是能让人快速理解、轻松维护、容易扩展的。

相信大家都有过维护别人代码的经历，如果各有各的风格，而没有遵循一定的规范和约定的话，那真的是挺痛苦的一件事。当然，既然编写代码被称作是一种艺术，那难免会有多样性。所以这里不会有太多「极端」的要求，只是提出一些建议和判断标准。

怎样提升代码可读性

这里主要从三个方面说明如何提升代码的可读性：

- 表面层次的改进。
- 简化逻辑。
- 重新组织代码。

从表面层次改进

表面层次的改进是指：选择合适的名字、写清晰的注释、将代码整理为更好的格式等等很容易应用的方式。

选择合适的名字

当我们在代码中给方法、变量等命名的时候，应该遵循「将信息装入名字 」这一原则。

要将信息装入名字就需要我们在命名时选择专业的词，避免空洞、泛泛的词。比如，单字母、tmp、buf 等无意义的词。当然在循环中大家已经习惯了用 i, j 等来表示索引，所以在这里也可以延续习惯。

写清晰的注释

注释应该是说明代码的意图，而不是简单的复述代码的行为。当我们在写注释时，应当是从更高的思维层次上来说明编写这段代码时的想法，就像是一个作家在阐述自己写作时的想法一样。 比如：

```
1 // 对于这些数据，二叉树要比哈希表快得多。
```

千万不要只是写一大段谁都能从代码里看出来的废话。

更好的代码风格

代码宽度

目前主流的代码规范都推荐代码宽度保持在 80 为宜，这么做当然是有历史原因，但在现在也还是有其实用价值的。因为将代码宽度限制在 80，是在需要打印代码的时候，完美适配 A4 纸的宽度。即使只是将代码贴在个人博客或在线网站上，这也是最适合代码阅读的宽度。当使用大屏显示器编程时，这个宽度也是很适合分屏工作的。

列对齐÷在这里笔者不推荐使用「列对齐」，比如：

```
var name     = "name";
var location = "location";
var phone    = "phone";
var url      = "url";
```

因为列对齐看起来确实还不错，让代码的阅读更轻松了些，但这样建立和维护对齐的工作量很大，当某一行有了些细微的变动，其他很多行也要跟着动，而且大部分都还只是空白。当然，如果你觉得这样做的工作量还可以接受，也是可以试一试的。

代码顺序

在组织方法的时候，应该遵循一定的逻辑顺序。但具体要遵照什么逻辑顺序，是可以按照自己的想法的，比如，从「重要」到「不重要」、按字母顺序排序等等。但最重要的是要一直坚持已有的风格，不一致的风格比没有风格更让人混乱。

这里也介绍一些好的代码风格：

- Google 的代码风格：包含 C++、Object-C、Java、Python、R、Shell、HTML\CSS、JavaScript、AngularJS、Common Lisp 和 Vimscript。
- Feross JavaScript 代码规范。

简化逻辑

每当你看到一个复杂的逻辑、一个巨大的表达式、一大堆变量时，你就应该思考应该怎么优化它们。因为这些都会增加你头脑的压力，要知道每个人的短期记忆都是很有限的。当你不得不思考过多的事情时，很可能在不知不觉中就产生 bug。 这里介绍几个简化逻辑的方法：

- 拆分复杂表达式。要拆分复杂表达式，可以使用「解释变量」的方法。比如：

  ```
  if line.split(':')[0].strip() == "root"
  ```

  这里我们可以加入一个额外的解释变量：

  ```
  username = line.split(':')[0].strip()
  if username == "root"
  ...
  ```

- 德摩根定理。 如果你学过「电子电路」这门课，那你一定对德摩根定理有印象。也就是对于一个布尔表达式，有两种等价写法：
  - (not a) and (not b) and (not c) = not (a or b or c)
  - (not a) or (not b) or (not c) = not (a and b and c)

你可以用德摩根定律来让你的布尔表达式更具有可读性。

- 尽可能简化变量。对于变量的使用，主要有三个需要注意的地方：
  - 变量越多，越难全部追踪它们的动向。
  - 变量作用域越大，就需要跟踪它们的动向越久。
  - 变量改变得越频繁，就越难跟踪其当前值。

重新组织代码

重新组织代码俗称「重构」，没有把握时不要轻易使用。这里介绍几个简单常用的方法，进阶的话可以去看《重构：改善既有代码的设计》这本书。

这里简单说说三种组织代码的方法：

- 抽取不相关的子问题。也就是要积极发现并抽取出不相关的子逻辑，在看某个代码块时，问问自己这段代码能不能抽取到独立的函数中。要知道「所谓工程学就是把大问题拆分成小问题再把这些问题的解决方案放回一起」，在编程过程中我们也应当经常拆分大问题，这能让你的代码更健壮和易读。
- 一次只做一件事。同时在做几件事的代码会让人很难理解，一个代码块可能会做初始化对象、解析输入、在界面展示。过多的不同业务代码纠缠在一起，会让你很难清楚理解它的逻辑。因此，我们应当将代码组织的尽可能清晰，我们不仅可以将一个大函数拆分为多个小一些的函数来区分边界，也可以在函数内部组织代码，使得其感觉上像是有分开的逻辑段，就像下面这张图片展示的一样。

图片来自于「编写可读代码的艺术」

- 把想法变成代码。其实检验代码可读性最简单的方法就是看看自己能不能很简单的将代码解释给别人。当人在解释一件复杂的事情时，最容易发现自己没有完全理解的小细节。能够用「自然语言」解释想法是很有价值的一个能力，这需要把一个想法归纳成精炼的概念，这不仅能帮助他人理解，也有助于自己把这个想法想得更清晰。

  这里介绍一个简单的过程来锻炼你的这项能力：

  1. 像对着一个同事一样用自然语言描述代码要做什么。
  2. 注意描述中所用的关键词和短语。
  3. 尽可能写出与描述相匹配的代码。
     当然看起来很简单，但其实是需要很大量练习的，更重要的是要有这方面的意识。