02-从哪些维度评判代码质量的好坏？如何具备写出高质量代码的能力？

在我的工作经历中，每当同事评论起项目代码质量的时候，听到的最多的评语就是：“代码写得很烂”或者“代码写得很好”。用“好”“烂”这样的字眼来描述，非常地笼统。当我具体问到底如何烂、如何好的时候，尽管大部分同事都能简单地罗列上几个点，但往往都不够全面、非常零碎，也切不中要害。

当然，也有一些工程师对如何评价代码质量有所认识，比如，好代码是易扩展、易读、简单、易维护的等等，但他们对于这些评价的理解往往只停留在表面概念上，对于诸多更深入的问题，比如，“怎么才算可读性好？什么样的代码才算易扩展、易维护？可读、可扩展与可维护之间有什么关系？可维护中‘维护’两字该如何理解？”等等，并没有太清晰的认识。

对于程序员来说，辨别代码写得“好”还是“烂”，是一个非常重要的能力。这也是我们写出好代码的前提。毕竟，如果我们连什么是好代码、什么是烂代码，都分辨不清，又谈何写出好代码呢？

所以，今天我们就聊一聊关于代码质量评判的相关问题，希望你在学完今天的内容之后，对代码质量的评判有个更加清晰、更加透彻的认识和理解。

如何评价代码质量的高低？

实际上，咱们平时嘴中常说的“好”和“烂”，是对代码质量的一种描述。“好”笼统地表示代码质量高，“烂”笼统地表示代码质量低。对于代码质量的描述，除了“好”“烂”这样比较简单粗暴的描述方式之外，我们也经常会听到很多其他的描述方式。这些描述方法语义更丰富、更专业、更细化。我搜集整理了一下，罗列在了下面。这些几乎涵盖我们所能听到的描述代码质量的所有常用词汇，你可以看一看。

> 灵活性（flexibility）、可扩展性（extensibility）、可维护性（maintainability）、可读性（readability）、可理解性（understandability）、易修改性（changeability）、可复用（reusability）、可测试性（testability）、模块化（modularity）、高内聚低耦合（high cohesion loose coupling）、高效（high effciency）、高性能（high performance）、安全性（security）、兼容性（compatibility）、易用性（usability）、整洁（clean）、清晰（clarity）、简单（simple）、直接（straightforward）、少即是多（less code is more）、文档详尽（well-documented）、分层清晰（well-layered）、正确性（correctness、bug free）、健壮性（robustness）、鲁棒性（robustness）、可用性（reliability）、可伸缩性（scalability）、稳定性（stability）、优雅（elegant）、好（good）、坏（bad）……

看到如此多的描述词，你可能要问了，我们到底该用哪些词来描述一段代码的质量呢？

实际上，我们很难通过其中的某个或者某几个词汇来全面地评价代码质量。因为这些词汇都是从不同维度来说的。这就好比，对于一个人的评价，我们需要综合各个方面来给出，比如性格、相貌、能力、财富等等。代码质量高低也是一个综合各种因素得到的结论。我们并不能通过单一的维度去评价一段代码写的好坏。比如，即使一段代码的可扩展性很好，但可读性很差，那我们也不能说这段代码质量高。

除此之外，不同的评价维度也并不是完全独立的，有些是具有包含关系、重叠关系或者可以互相影响的。比如，代码的可读性好、可扩展性好，就意味着代码的可维护性好。而且，各种评价维度也不是非黑即白的。比如，我们不能简单地将代码分为可读与不可读。如果用数字来量化代码的可读性的话，它应该是一个连续的区间值，而非0、1这样的离散值。

不过，我们真的可以客观地量化一段代码质量的高低吗？答案是否定的。对一段代码的质量评价，常常有很强的主观性。比如，怎么样的代码才算可读性好，每个人的评判标准都不大一样。这就好比我们去评价一本小说写得是否精彩，本身就是一个很难量化的、非常主观的事情。

正是因为代码质量评价的主观性，使得这种主观评价的准确度，跟工程师自身经验有极大的关系。越是有经验的工程师，给出的评价也就越准确。相反，资历比较浅的工程师就常常会觉得，没有一个可执行的客观的评价标准作为参考，很难准确地判断一段代码写得好与坏。有的时候，自己觉得代码写得已经够好了，但实际上并不是。所以，这也导致如果没有人指导的话，自己一个人闷头写代码，即便写再多的代码，代码能力也可能一直没有太大提高。

最常用的评价标准有哪几个？

仔细看前面罗列的所有代码质量评价标准，你会发现，有些词语过于笼统、抽象，比较偏向对于整体的描述，比如优雅、好、坏、整洁、清晰等；有些过于细节、偏重方法论，比如模块化、高内聚低耦合、文档详尽、分层清晰等；有些可能并不仅仅局限于编码，跟架构设计等也有关系，比如可伸缩性、可用性、稳定性等。

为了做到有的放矢、有重点地学习，我挑选了其中几个最常用的、最重要的评价标准，来详细讲解，其中就包括：可维护性、可读性、可扩展性、灵活性、简洁性（简单、复杂）、可复用性、可测试性。接下来，我们逐一讲解一下。

1.可维护性（maintainability）

我们首先来看，什么是代码的“可维护性”？所谓的“维护代码”到底包含哪些具体工作？

落实到编码开发，所谓的“维护”无外乎就是修改bug、修改老的代码、添加新的代码之类的工作。所谓“代码易维护”就是指，在不破坏原有代码设计、不引入新的bug的情况下，能够快速地修改或者添加代码。所谓“代码不易维护”就是指，修改或者添加代码需要冒着极大的引入新bug的风险，并且需要花费很长的时间才能完成。

我们知道，对于一个项目来说，维护代码的时间远远大于编写代码的时间。工程师大部分的时间可能都是花在修修bug、改改老的功能逻辑、添加一些新的功能逻辑之类的工作上。所以，代码的可维护性就显得格外重要。

维护、易维护、不易维护这三个概念不难理解。不过，对于实际的软件开发来说，更重要的是搞清楚，如何来判断代码可维护性的好坏。

实际上，可维护性也是一个很难量化、偏向对代码整体的评价标准，它有点类似之前提到的“好”“坏”“优雅”之类的笼统评价。代码的可维护性是由很多因素协同作用的结果。代码的可读性好、简洁、可扩展性好，就会使得代码易维护；相反，就会使得代码不易维护。更细化地讲，如果代码分层清晰、模块化好、高内聚低耦合、遵从基于接口而非实现编程的设计原则等等，那就可能意味着代码易维护。除此之外，代码的易维护性还跟项目代码量的多少、业务的复杂程度、利用到的技术的复杂程度、文档是否全面、团队成员的开发水平等诸多因素有关。

所以，从正面去分析一个代码是否易维护稍微有点难度。不过，我们可以从侧面上给出一个比较主观但又比较准确的感受。如果bug容易修复，修改、添加功能能够轻松完成，那我们就可以主观地认为代码对我们来说易维护。相反，如果修改一个bug，修改、添加一个功能，需要花费很长的时间，那我们就可以主观地认

为代码对我们来说不易维护。

你可能会说，这样的评价方式也太主观了吧？没错，是否易维护本来就是针对维护的人来说的。不同水平的人对于同一份代码的维护能力并不是相同的。对于同样一个系统，熟悉它的资深工程师会觉得代码的可维护性还不错，而一些新人因为不熟悉代码，修改bug、修改添加代码要花费很长的时间，就有可能会觉得代码的可维护性不那么好。这实际上也印证了我们之前的观点：代码质量的评价有很强的主观性。

2.可读性（readability）

软件设计大师Martin Fowler曾经说过：“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”翻译成中文就是：“任何傻瓜都会编写计算机能理解的代码。好的程序员能够编写人能够理解的代码。”Google内部甚至专门有个认证就叫作Readability。只有拿到这个认证的工程师，才有资格在code review的时候，批准别人提交代码。可见代码的可读性有多重要，毕竟，代码被阅读的次数远远超过被编写和执行的次数。

我个人认为，代码的可读性应该是评价代码质量最重要的指标之一。我们在编写代码的时候，时刻要考虑到代码是否易读、易理解。除此之外，代码的可读性在非常大程度上会影响代码的可维护性。毕竟，不管是修改bug，还是修改添加功能代码，我们首先要做的事情就是读懂代码。代码读不大懂，就很有可能因为考虑不周全，而引入新的bug。

既然可读性如此重要，那我们又该如何评价一段代码的可读性呢？

我们需要看代码是否符合编码规范、命名是否达意、注释是否详尽、函数是否长短合适、模块划分是否清晰、是否符合高内聚低耦合等等。你应该也能感觉到，从正面上，我们很难给出一个覆盖所有评价指标的列表。这也是我们无法量化可读性的原因。

实际上，code review是一个很好的测验代码可读性的手段。如果你的同事可以轻松地读懂你写的代码，那说明你的代码可读性很好；如果同事在读你的代码时，有很多疑问，那就说明你的代码可读性有待提高了。

3.可扩展性（extensibilty）

可扩展性也是一个评价代码质量非常重要的标准。它表示我们的代码应对未来需求变化的能力。跟可读性一样，代码是否易扩展也很大程度上决定代码是否易维护。那到底什么是代码的可扩展性呢？

代码的可扩展性表示，我们在不修改或少量修改原有代码的情况下，通过扩展的方式添加新的功能代码。说直白点就是，代码预留了一些功能扩展点，你可以把新功能代码，直接插到扩展点上，而不需要因为要添加一个功能而大动干戈，改动大量的原始代码。

关于代码的扩展性，在后面讲到“对修改关闭，对扩展开放”这条设计原则的时候，我会来详细讲解，今天我们只需要知道，代码的可扩展性是评价代码质量非常重要的标准就可以了。

4.灵活性（flexibility）

灵活性也是描述代码质量的一个常用词汇。比如我们经常会听到这样的描述：“代码写得很灵活”。那这里的“灵活”该如何理解呢？

尽管有很多人用这个词汇来描述代码的质量。但实际上，灵活性是一个挺抽象的评价标准，要给灵活性下个定义也是挺难的。不过，我们可以想一下，什么情况下我们才会说代码写得好灵活呢？我这里罗列了几个场

景，希望能引发你自己对什么是灵活性的思考。

- 当我们添加一个新的功能代码的时候，原有的代码已经预留好了扩展点，我们不需要修改原有的代码，只要在扩展点上添加新的代码即可。这个时候，我们除了可以说代码易扩展，还可以说代码写得好灵活。
- 当我们要实现一个功能的时候，发现原有代码中，已经抽象出了很多底层可以复用的模块、类等代码，我们可以拿来直接使用。这个时候，我们除了可以说代码易复用之外，还可以说代码写得好灵活。
- 当我们使用某组接口的时候，如果这组接口可以应对各种使用场景，满足各种不同的需求，我们除了可以说接口易用之外，还可以说这个接口设计得好灵活或者代码写得好灵活。

从刚刚举的场景来看，如果一段代码易扩展、易复用或者易用，我们都可以称这段代码写得比较灵活。所以，灵活这个词的含义非常宽泛，很多场景下都可以使用。

5.简洁性（simplity）

有一条非常著名的设计原则，你一定听过，那就是KISS原则：“Keep It Simple，Stupid”。这个原则说的意思就是，尽量保持代码简单。代码简单、逻辑清晰，也就意味着易读、易维护。我们在编写代码的时候，往往也会把简单、清晰放到首位。

不过，很多编程经验不足的程序员会觉得，简单的代码没有技术含量，喜欢在项目中引入一些复杂的设计模式，觉得这样才能体现自己的技术水平。实际上，**思从深而行从简，真正的高手能云淡风轻地用最简单的方法解决最复杂的问题。这也是一个编程老手跟编程新手的本质区别之一。**

除此之外，虽然我们都能认识到，代码要尽量写得简洁，符合KISS原则，但怎么样的代码才算足够简洁？不是每个人都能很准确地判断出来这一点。所以，在后面的章节中，当我们讲到KISS原则的时候，我会通过具体的代码实例，详细给你解释，“为什么KISS原则看似非常简单、好理解，但实际上用好并不容易”。今天，我们就暂且不展开详细讲解了。

6.可复用性（reusability）

代码的可复用性可以简单地理解为，尽量减少重复代码的编写，复用已有的代码。在后面的很多章节中，我们都会经常提到“可复用性”这一代码评价标准。

比如，当讲到面向对象特性的时候，我们会讲到继承、多态存在的目的之一，就是为了提高代码的可复用性；当讲到设计原则的时候，我们会讲到单一职责原则也跟代码的可复用性相关；当讲到重构技巧的时候，我们会讲到解耦、高内聚、模块化等都能提高代码的可复用性。可见，可复用性也是一个非常重要的代码评价标准，是很多设计原则、思想、模式等所要达到的最终效果。

实际上，代码可复用性跟DRY（Don’t Repeat Yourself）这条设计原则的关系挺紧密的，所以，在后面的章节中，当我们讲到DRY设计原则的时候，我还会讲更多代码复用相关的知识，比如，“有哪些编程方法可以提高代码的复用性”等。

7.可测试性（testability）

相对于前面六个评价标准，代码的可测试性是一个相对较少被提及，但又非常重要的代码质量评价标准。代码可测试性的好坏，能从侧面上非常准确地反应代码质量的好坏。代码的可测试性差，比较难写单元测试，那基本上就能说明代码设计得有问题。关于代码的可测试性，我们在重构那一部分，会花两节课的时间来详细讲解。现在，你暂时只需要知道，代码的可测试性非常重要就可以了。

如何才能写出高质量的代码？

我相信每个工程师都想写出高质量的代码，不想一直写没有成长、被人吐槽的烂代码。那如何才能写出高质量的代码呢？针对什么是高质量的代码，我们刚刚讲到了七个最常用、最重要的评价指标。所以，问如何写出高质量的代码，也就等同于在问，如何写出易维护、易读、易扩展、灵活、简洁、可复用、可测试的代码。

要写出满足这些评价标准的高质量代码，我们需要掌握一些更加细化、更加能落地的编程方法论，包括面向对象设计思想、设计原则、设计模式、编码规范、重构技巧等。而所有这些编程方法论的最终目的都是为了编写出高质量的代码。

比如，面向对象中的继承、多态能让我们写出可复用的代码；编码规范能让我们写出可读性好的代码；设计原则中的单一职责、DRY、基于接口而非实现、里式替换原则等，可以让我们写出可复用、灵活、可读性好、易扩展、易维护的代码；设计模式可以让我们写出易扩展的代码；持续重构可以时刻保持代码的可维护性等等。具体这些编程方法论是如何提高代码的可维护性、可读性、可扩展性等等的呢？我们在后面的课程中慢慢来学习。

重点回顾

今天的内容到此就讲完了。我们来一起回顾一下，你需要重点掌握的几个知识点。

1.如何评价代码质量的高低？

代码质量的评价有很强的主观性，描述代码质量的词汇也有很多，比如可读性、可维护性、灵活、优雅、简洁等，这些词汇是从不同的维度去评价代码质量的。它们之间有互相作用，并不是独立的，比如，代码的可读性好、可扩展性好就意味着代码的可维护性好。代码质量高低是一个综合各种因素得到的结论。我们并不能通过单一的维度去评价一段代码的好坏。

2.最常用的评价标准有哪几个？

最常用到几个评判代码质量的标准是：可维护性、可读性、可扩展性、灵活性、简洁性、可复用性、可测试性。其中，可维护性、可读性、可扩展性又是提到最多的、最重要的三个评价标准。

3.如何才能写出高质量的代码？

要写出高质量代码，我们就需要掌握一些更加细化、更加能落地的编程方法论，这就包含面向对象设计思想、设计原则、设计模式、编码规范、重构技巧等等，这也是我们后面课程学习的重点。

课堂讨论

除了我今天提到的这些，你觉得还有哪些其他的代码评价标准非常重要？聊一聊你心目中的好代码是什么样子的？

欢迎在留言区发表你的观点，积极参与讨论。你也可以把这篇文章分享给你的朋友，邀请他一起学习。

精选留言：

- 丁丁历险记 2019-11-04 18:14:18

笔记：
个人理解，代码质量评判
1 机器的运行效率 （往往还和可读性相冲突，但又非绝对冲突） ， 有时候算法在没优化好的时候，时间空间是可以一起省下来的。
2 代码可管理行数。 好的代码，层次分明，职能分明，让人感受到代码品味。

2 最常用的评价标准。
这块我一直没细分好，经常和同事开玩笑说“代码品味”（尊重大脑的特性，写出可便于维护的代码。程序 = 数据结构+算法 算法分control 相关和logic 相关。 合理的把control 相关与logic 相关进行分离就是非常好的套路 ，时间久了，看到违和感重的代码就很敏感了，主要是要求别把代码写死写散， 像dry 等基础原则都没遵守的，烂用全局变量的，创建对象没用框架的createObj 的，没支持依赖注入的，直接code review 时会指出 ）
如何才能写出高质量的代码?
做中学。 我只说我自己，我在每完成一份工作后，都要拿出很大一部分时间来优化重构，自己改自己的代码。 【主要套路来源，代码整洁之道，重构-改善即有代码设计】 这算是我自律的一部分，我很珍惜工作中的开发实践。纯理论的东西学多了，人会飘飘乎乎的，需要实操来落地。
[26赞]

- AF 2019-11-04 19:12:05
  “思从深而行从简，真正的高手能云淡风轻地用最简单的方法解决最复杂的问题。” —— 这句话是写代码的精髓 [11赞]

- burning 微信超級會員 2019-11-04 19:32:54
  稳定性很重要 尤其在前后端分离开发时。说好了按约定的接口规则开发 可联调时各种出错 甚至接口崩溃报异常。前端成测试了 [5赞]

- 时光勿念 2019-11-05 07:23:28
  好看的代码千篇一律，垃圾的代码花样百出 [3赞]

- lijun 2019-11-04 21:56:37
  Jdk源码和spring源码写的非常棒，可惜目前还看不懂。 [2赞]

- 爱吃彩虹糖的猫~ 2019-11-04 20:10:55
  001 如何评价代码质量的高低?
  代码质量的评价有很强的主观性，描述代码质量的词汇也有很多，比如可读性、可维护性、灵活、优雅、简洁等。
  002 常用的评价标准
  可维护性、可读性、可扩展性、灵活性、简洁性、可复用性、可测试性。
  003 如何才能写出高质量的代码?
  需要掌握一些更加细化、更加能落地的编程方法论，这就包含面向对象设计思想、设计原则、设计模式、编码规范、重构技巧等等
  [2赞]

- 编程界的小学生 2019-11-05 12:56:36
  可扩展，可读性，健壮性
  我认为mybatis做的很到位，小巧简单，功能强大。健壮稳定可扩展。正是有了这么好的代码，才使得有很多框架可以无缝集成进来，比如spring-mybatis，再比如sharding-jdbc [1赞]

- 我能走多远 2019-11-05 07:53:57

代码健壮性-异常场景的处理。代码可维护性还包含日志记录。可定位性。 [1赞]

- 仙道 2019-11-04 23:32:24
  我觉得最好是能把注释尽可能的写详细，最好能举几个例子。
  因为员工之间水平层次不齐，哪怕是再好的代码，在他眼里就是垃圾
  遇到爱扯皮的同事，真的很难受 [1赞]

- 赌神很低调 2019-11-04 21:26:49
  好代码就像一篇好文章，层次分明，用词贴切，简洁素雅，形象化抽象，脉络化复杂，而且有趣，吸引你通宵看完，然后合上书本，意犹未尽。 [1赞]

- SweetyTang 2019-11-04 18:38:21
  争哥，好的代码是不是也得考虑错误处理 [1赞]

- Sisyphus235 2019-11-05 13:13:30
  代码不要过度设计，根据需求迭代，留有接口和扩展性。同时大型项目一定要考虑安全性，做好过滤和校检。

- 张瑞浩 2019-11-05 13:10:28
  深刻同意，简单有效是代码的最终极答案。
  之前负责项目中，有些业务代码刻意使用设计模式，比如访问者模式，导致后来者很难阅读；另外递归这种尽量少在业务代码中使用，第一个是阅读性，第二是问题可维护性

- 为你而来 2019-11-05 12:49:19
  我认为的好的代码：当你写的代码，交给别人时，他很容易看懂，能够快速理解你想要表达的意思；很方便的进行测试，以理解各个模块的功能；容易扩展，添加新功能时，不出bug，不重复造轮子，可以复用你的代码。

  怎么做到呢？ 遵循设计模式、设计原则、有测试模块、编码规范（命名、模块化、统一编码风格）、有文档。

- helloworld 2019-11-05 12:28:19
  思从深而行从简，说的真好，写程序如此，做人亦如此！

- 陈华应 2019-11-05 12:13:19
  思从深，行从简！赞！

- 胖大海 2019-11-05 11:16:06
  对于一些特殊用途的业务需求，可能代码还需要考虑性能的一些硬性指标，因此就有可能需要牺牲一部分可读性可扩展性等；对于一些特殊的行业，比如医药行业的支持系统，还有可能需要考虑一些合规性政策对代码编写上的要求， 这方面恰恰是我们程序员们不太愿意去关注的。

- 马小平 2019-11-05 10:55:52
  好的代码都差不多，不好的代码各有各的不好

- 天空只能仰望？ 2019-11-05 10:55:10
  代码执行效率，代码稳定性两个重要指标，工作中很多公司都很重视这两个指标，都进行压测或者全链路压测去检验

- 小伟 2019-11-05 10:53:41

  代码除了运行外(给机器看)，大部分时间是给人看的。

  一份代码是“好”还是“坏”，首先要能让人读懂，所以可读性是首位的，甚至牺牲某些性能也要保证可读性。读代码前，了解这份代码的背景很重要，说白话就是要知道这段代码是在干嘛，故一段overall的注释很有必要和重要，可以是思路、算法步骤或功能说明。最好是把可读变成易读。王老师的一句话很好，你的同事或其他人能轻松读懂你的代码，那么这个代码就是“好”代码。

  其次，要有代码设计要有前瞻性，说白话就是业务逻辑分支要尽量多考虑到。业务逻辑决定了采用哪些设计模式来达成易扩展和易复用。如果做不到考虑周全，那么尽量选择那些有插件接口的模式来规提高易扩展和易复用性，如策略、工厂、模板等。

  最后，易测试非常重要，尤其是在CI/CD的模式下，代码要对自动化的单元、集成、回归等测试友好，否则CI/CD很难有效率和效果。具体做法可参考各种DD，如TDD、BDD、DDD等。

  总结，代码先要能读懂、好读懂，再能扩展、能复用，最后好测试、易测试。对于“好”代码，业务逻辑对错，优先级最低。可先做好前面几步，再调试业务逻辑，而不是反过来，这样的效率和效果都比较高。

  个人看法，欢迎拍砖。