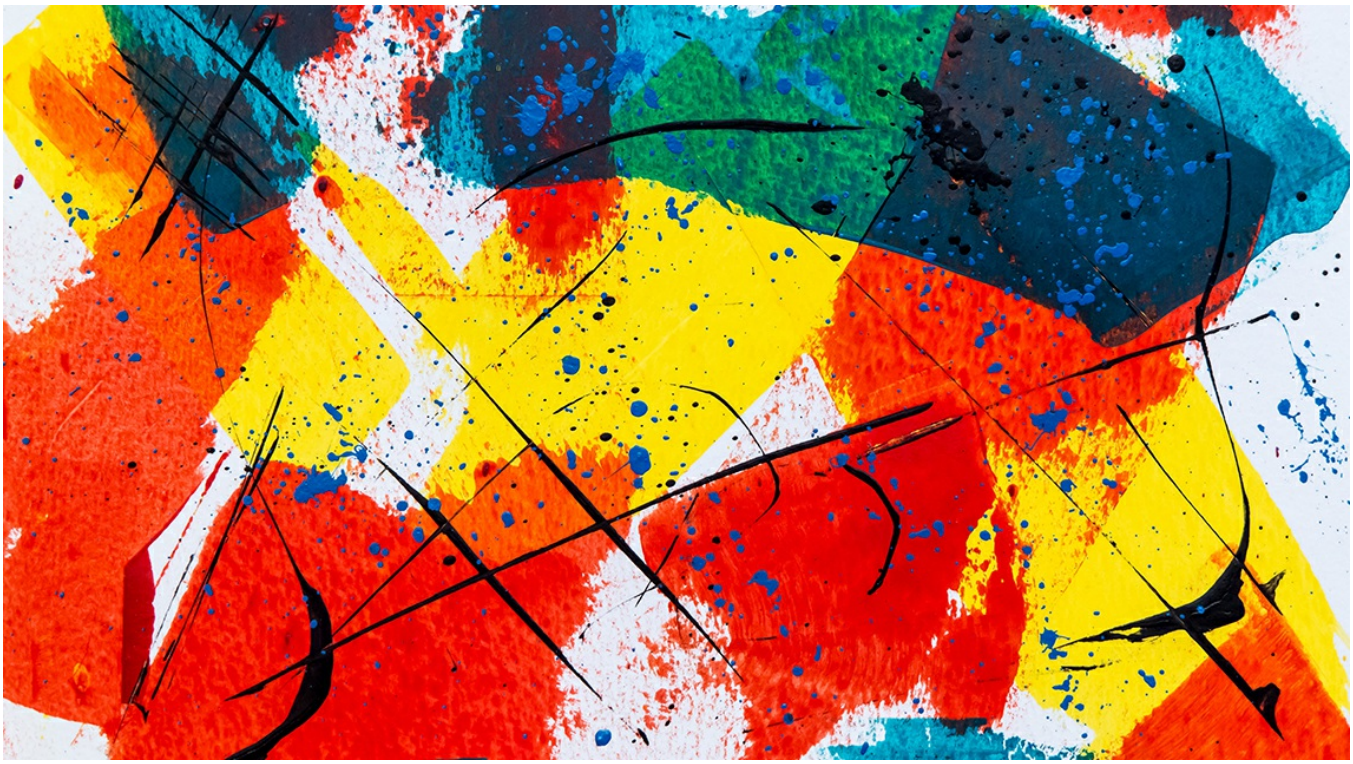
01 | 从条件运算符说起，反思什么是好代码

2019-01-04 范学雷



写出优秀的代码是我们每一个程序员的毕生追求，毕竟写代码本身就是个技术活，代码的好坏，其实也就是我们工艺的好坏。作为一个技术类的工种，我们没有理由不去思考如何写出优秀、让人惊叹的代码。

那什么样的代码才是优秀的代码呢？对于这个问题，我想每个人心中都会有自己的答案。今天我就来和你聊聊我的思考。

对于条件运算符（?:）的使用，我估摸着你看到过相关的争论，或者自己写代码的时候也不知道到底该不该使用条件运算符，或者什么情况下使用？这些微不足道的小话题随时都可以挑起激烈的争论。

C语言之父丹尼斯·里奇就属于支持者。在《C程序设计语言》这本书里，他使用了大量简短、直观的条件运算符。

然而还有一些人，对条件运算符完全丧失了好感，甚至把“永远不要使用条件运算符”作为一条C语言高效编程的重要技巧。

比如说吧，下面的这个例子，第一段代码使用条件语句，第二段代码使用条件运算符。 你觉得哪一段代码更“优秀”呢？

```
if (variable != null) {
   return variable.getSomething();
}

return null;
```

```
return variable != null ? variable.getSomething() : null;
```

同样使用条件运算符，你会喜欢下面代码吗？

```
return x >= 90 ? "A" : x >= 80 ? "B" : x >= 70 ? "C" : x >= 60 ? "D" : "E";
```

十多年前，作为一名C语言程序员，我非常喜欢使用条件运算符。因为条件运算符的这种压缩方式，使代码看起来简短、整洁、干净。 而且，如果能把代码以最少的行数、最简短的方式表达出来，心里也颇有成就感。

后来，我的一位同事告诉我，对于我使用的条件运算符的部分代码，他要仔细分析才知道这一小行代码想要表达的逻辑，甚至有时候还要翻翻书、查查操作符的优先级和运算顺序，拿笔画一画逻辑关系，才能搞清楚这一小行代码有没有疏漏。

这么简单的代码，为什么还要确认运算符的优先级和运算顺序呢？因为只是“看起来”对的代码，其实特别容易出问题。所以，一定要反复查验、确认无误才能放心。

这么简单的代码，真的需要这么认真检查吗？超级简单的代码的错误，往往是我们最容易犯的一类编码错误。我个人就是犯过很多次这种低级、幼稚的错误，并且以后一定还会再犯。比如下面的这段有问题的代码，就是我最近犯的一个非常低级的代码错误：

```
// Map for debug logging.  Enable debug log if SSLLogger is on.
private final Map<Integer, byte[]> logMap =
        SSLLogger.isOn ? null : new LinkedHashMap<>();
```

正确的代码应该是：

```
// Map for debug logging. Enable debug log if SSLLogger is on.
private final Map<Integer, byte[]> logMap =
        SSLLogger.isOn ? new LinkedHashMap<>() : null;
```

你可能会说，这个代码错误看起来太幼稚、太低级、太可笑了吧？ 确实是这样的。这段错误的代码，我的眼睛不知道看过了它们多少次，可是这个小虫子（bug）还是华丽丽地逃脱了我的注意，进入了**JDK 11的最终发布版**。

如果使用条件语句，而不是条件运算符，这个幼稚错误发生的概率会急剧下降。**坚持使用最直观的编码方式，而不是追求代码简短，真的可以避免很多不必要的错误**。所以说啊，选择适合的编码方式，强调代码的检查、评审、校验，真的怎么都不算过分。

现在，如果你要再问我喜欢哪种编码方式，毫无疑问，我喜欢使用条件语句，而不是条件运算符。因为，用条件语句这种编码方式，可以给我确定感，我也不需要挑战什么高难度动作；而看代码的人，也可以很确定，很轻松，不需要去查验什么模糊的东西。

这种阅读起来的确定性至少有三点好处，第一点是可以减少代码错误；第二点是可以节省我思考的时间；第三点是可以节省代码阅读者的时间。

减少错误、节省时间，是我们现在选择编码方式的一个最基本的原则。

《C程序设计语言》这本C程序员的圣经，初次发表于1978年。那个年代的代码，多数很简单直接。简短的代码，意味着节省昂贵的计算能力，是当时流行的编码偏好。而现在，计算能力不再是瓶颈，如何更高效率地开发复杂的软件，成了我们首先需要考虑的问题。

有一些新设计的编程语言，不再提供条件运算符。 比如，Kotlin语言的设计者认为，编写简短的代码绝对不是Kotlin的目标。所以，Kotlin不支持条件运算符。 Go语言的设计者认为，条件运算符的滥用，产生了许多难以置信的、难以理解的复杂表达式。所以，Go语言也不支持条件运算符。

我们看到，**现实环境的变化，影响着我们对于代码“好”与“坏”的判断标准。**

“好”的代码与“坏”的代码

虽然对于“什么是优秀的代码“难以形成一致意见，但是这么多年的经验，让我对代码“好”与“坏”积累了一些自己的看法。

比如说，“好”的代码应该：

1. 容易理解；
2. 没有明显的安全问题；
3. 能够满足最关键的需求；
4. 有充分的注释；

5. 使用规范的命名；
6. 经过充分的测试。

“坏”的代码包括：

1. 难以阅读的代码；
2. 浪费大量计算机资源的代码；
3. 代码风格混乱的代码；
4. 复杂的、不直观的代码；
5. 没有经过适当测试的代码。

当然，上面的列表还可以很长很长，长到一篇文章都列不完、长到我们都记不住的程度。

优秀的代码是“经济”的代码

大概也没人想记住这么多条标准吧？所以，**关于优秀代码的特点，我想用“经济”这一个词语来表达**。这里的“经济”，指的是使用较少的人力、物力、财力、时间、空间，来获取较大的成果或收益 。或者简单地说，**投入少、收益大、投资回报高**。为了方便，你也可以先理解为节俭或者抠门儿的意思。

当然，使用一个词语表达肯定是以偏概全的。但是，比起一长串的准则，一个关键词的好处是，更容易让人记住。我想这点好处可以大致弥补以偏概全的损失。

该怎么理解“经济”呢？这需要我们把代码放到软件的整个生命周期里来考察。

关于软件生命周期，我想你应该很熟悉了，我们一起来复习一下。一般而言，一个典型的软件生命周期，大致可以划分计划、分析和设计、代码实现、测试、运营和维护这六个阶段。在软件维护阶段，可能会有新的需求出现、新的问题产生、旧问题的浮现，这些因素可能就又要推动新一轮的计划，分析、设计、实现、测试、运营。这样，这个周期就会反复迭代，反复的循环，像一个周而复始的流水线。

当我们说投入少的时候，说的是这整个生命周期，甚至是这个周而复始的生命周期的投入少。比如说，代码写得快，可是测试起来一大堆问题，就不是经济的。

现代的大型软件开发，一般都会有比较细致的分工，在各个阶段参与的人是不同的；甚至在相同的阶段，也会有多人参与。一个稍有规模的软件，可能需要数人参与设计和实现。而为了使测试相对独立，软件测试人员和软件实现人员也是相对独立的，而且他们具备不同的优势和技能。

所以，当我们考虑投入的时候，还要考虑这个生命周期里所有的参与人员。这些参与人员所处的立场、看问题的角度，所具有的资源禀赋，可能千差万别。比如说，如果客户需要阅读代码，才

知道系统怎么使用，就不是经济的。

是不是所有的软件都有这六个阶段呢？显然不是的，我本科的毕业论文程序，就完全没有运营和维护阶段，甚至也不算有测试阶段。我当时的毕业论文是一个关于加快神经网络学习的数学算法。只要验证了这个算法收缩得比较快，程序的使命就完成了，程序就可以退出销毁了。 所以，运营和维护阶段，甚至测试阶段，对当时的我而言，都是不需要投入的阶段。

在现代商业社会里，尤其我们越来越倾向于敏捷开发、精益创业，提倡“快速地失败、廉价地失败”，很多软件走不到维护阶段就已经结束了。而且，由于人力资源的限制，当然包括资金的限制，一个程序员可能要承担很多种角色，甚至从开始有了想法，到软件实现结束，都是一个人在战斗，哪里分什么设计人员、测试人员。

对软件开发流程选择的差异，就带来了我们对代码质量理解，以及对代码质量重视程度的千差万别。 比如说，一个创业公司是万万不能照搬大型成熟软件的开发流程的。因为，全面的高质量、高可靠、高兼容性的软件可能并不是创业公司最核心的目标。如果过分纠缠于这些代码指标，创始人的时间、投资人的金钱可能都没有办法得到最有效的使用。

当然，越成熟的软件开发机制越容易写出优秀的代码。但是，**最适合当前现实环境的代码，才是最优秀的代码。**

所以，当我们考虑具体投入的时候，还要考虑我们所处的现实环境。 如果我们超出现实环境去讨论代码的质量，有时候会有失偏颇，丧失我们讨论代码质量的意义。

既然具体环境千差万别，那我们还有必要讨论什么是优秀的代码吗？优秀的代码还能有什么共同的规律吗？ 即使一个人做所有的事情，即使代码用完一次就废弃，我们长期积累下来的编写优秀代码的经验，依然可以帮助到很多人。

比如说，虽然创业公司的软件刚开始最核心的追求不是全面的高可靠性。可是，你也要明白，创业的目的不是为了失败，一旦创业公司稳住了阵脚，这个时候如果它们没有高可靠性的软件作为支撑，很快就会有反噬作用。 而程序员背锅，就是反噬的其中一个后果。

如何使用最少的时间、最少的资源，提供最可靠的软件，什么时候开始把可靠性提高到不可忽视的程度，有没有可能一开始就是高可靠的,这些就都是一个富有经验的创业公司技术负责人不得不考虑的问题。而我们总结出来的编写代码的经验，毫无疑问，可以为这些问题提供一些思路和出路。

为什么我们要从“经济”这个角度来衡量优秀的代码呢？ 因为这是一个可以让我们更加理性的概念。

一个营利性的公司，必须考虑投入产出比，没有人愿意做亏本的买卖，股东追求的是利润最大化。作为程序员，我们也必须考虑投入和产出。 首先，我们的产出必须大幅度大于公司对我们

的投入，否则就有随时被扫地出门的风险。然后，我们必须使用好我们的时间，在单位时间内创造更多的价值，否则，真的是没有功劳，只有徒劳。

编写代码的时候，如果遇到困惑或者两难，你要想一想，怎么做才能做到投资少、收益大？

即便具体环境千差万别，我还是有一些例子，可以和你一起分享：

1. 代码写得又快又好，是“经济”的；代码写得快，但是错误多，不是一个“经济”的行为。
2. 代码跑得又快又好，是“经济”的；代码跑得快，但是安全问题突出，不是一个“经济”的行为。
3. 代码写得精简易懂，是“经济”的；代码写得精简，但是没人看得懂，不是一个“经济”的行为。

总结

对于所有的程序员来说，每个人都会遇到两个有名的捣蛋鬼，一个捣蛋鬼是“合作”，另一个捣蛋鬼是“错误”。

要合作，就需要用大部分人都舒服的方式。程序员间合作交流最重要的语言便是代码，换句话说，这就需要我们规范地编写代码，使用大家都接受的风格。不规范的代码，我们可能节省了眼前的时间，但是测试、运营、维护阶段，就需要更多的时间。而一旦问题出现，这些代码会重新返工，又回到我们手里，需要阅读、修改，再一次浪费我们自己的时间。对于这些代码，每一点时间的付出，都意味着投入，意味着浪费，意味着我们损失了做更有意义事情的机会。

人人都会犯错误，代码都会有bug，可是有些错误的破坏力是我们无法承受的，其中，最典型的就是安全问题。很多安全问题对公司和个人造成不容忽视的恶劣影响。我见过因为安全问题破产的公司。这时候，甚至都不要谈什么投入产出比、经济效益了，所有的投入归零，公司破产，员工解散。这需要我们分外地卖力，拿出十二分的精神来处理潜在的安全威胁，编写安全的代码。

如果我们把规范和安全作为独立的话题，优秀的代码需要具备三个特征： 经济、规范、安全。这些内容就是我们接下来要在专栏里一起学习的主体。

好了，今天我们一口气聊了很多，主要是在探讨到底什么样的代码才是优秀的代码。这个问题你之前考虑过吗？和我今天讲的是否一样呢？欢迎你在留言区写写自己的想法，我们可以进一步讨论。也欢迎你把今天的文章分享给跟你协作的同学，看看你们之间的理解是否一致。

精选留言

郭解

13

多些实例

2019-01-04

编辑回复

您好，第一篇只是引子，后续内容会有很多正反面案例，请期待。

2019-01-04

Imperfect

22

有一句很经典的名言：当我写这行代码的时候，只有我和上帝知道这行代码的意思。一年之后...现在，就只有上帝知道了。

2019-01-05

作者回复

上帝很孤独.....

2019-01-05

W.J.Huang

13

个人认为，条件运算符适合简单的结果取舍，带有复杂的表达式或嵌套运算符的应该用if语句。另外，代码规范和语句检查，可以使用SonarQube. SonarQube不仅带来代码评审的便利，还可以直接安装在IDE, 比如Eclipse, 会时刻提醒不规范或不安全的代码，以提高开发人员提交代码质量。

2019-01-05

作者回复

，小伙伴们快来试试这个工具。 高手都在讨论区。

2019-01-05

秦凯

3

老师从软件生命周期来看代码好坏的角度让我对代码质量有了新的认识，好的代码应该是服务于整个软件生命周期的，或者说是服务于软件开发目标的，不能说代码中用了某个高深的技巧或技术就是好代码，应该跳出开发视角，从整个软件的目标思考，能服务于软件目标，适合软件目标的，对于整个软件生命周期都是经济的才是好的代码。

2019-01-06

作者回复

总结的特别棒、特别好。

要有意识地从最基础的概念开始理解一些问题，这就是我想通过这个专栏让你收获的东西之一。

2019-01-06

Change

3

以前觉得看不懂的代码就是厉害的有种不明觉厉的意思，看完老师的文章才觉得化繁为简通俗易懂的代码才是我们追求的好代码，其编写的过程是由简单到复杂，再由复杂到简单的一个转变。自己对安全这块也比较重视，期待老师的后续内容。

2019-01-04

作者回复

我们第三篇聊安全。 如果哪一天，你觉得复杂很简单，告诉我一下，我要恭喜你。

2019-01-05

Woj

3

好的代码是代码运行正常、bug很少、并且具有可读性和可维护性。一些企业自己有所有开发人员都必需遵守的编码规范，但是对于什么样的代码是最好的每个人的都有自己的标准、或者有太多的或太少的编码规则。这有多种原则和标准，例如，McCable 的复杂度度量。的确使用过多的编码标准和规则可能降低生产率和创造性。“同行评审”或“同事检查”代码分析工具等，都能用来检查问题或坚持标准。

2019-01-04

作者回复

标准多就像没标准。

2019-01-04

alan

2

谢谢老师，坚定了我认为代码是给人看的这个观念。

2019-01-04

作者回复

给人看，也给以后的自己看。

2019-01-04

yaya

2

一直觉得条件运算符复杂但是看起来漂亮，以后就安心用if语句了，命名这个问题是一个困扰我蛮久的问题，我英语水平就中等水平吧，可以读懂，看懂，但是自己写或者说就一般般，怎么好的来对一个函数命名呢，有什么比较好的规则吗？

2019-01-04

作者回复

后面我们会专门讨论命名的。

2019-01-04

南宁码王

2

代码范例太少啦

2019-01-04

作者回复

嗯，[]代码马上就来

2019-01-04

月城冥

1

请教一个具体的问题：java lambda该用还是不该用？个人感觉这个东西的可读性还是有点差的。什么样的场景该用，什么样的场景不该用？

2019-03-08

作者回复

lambda的可读性，主要是和我们通常的编码习惯不太一样。lambda在异步编程和函数编程方面，还是有可读性优势的。

我不是lambda的铁杆粉丝，主要是因为我做的工作大部分是非常基础的接口。lambda方式会产生大量的内部类和实例，这个有损效率。这个效率损耗如果发生在应用层，影响不大，可以忽略不计。可是要发生在使用频繁的底层接口，就是个不小的问题了。

2019-03-08

路过蜻蜓

1

我写代码时一般是先顺着写下来，然后再对需要重复使用的代码分离出去，写成一个方法，尽量将写在一个def里的代码简单点。

2019-01-11

作者回复

代码整理术！

2019-01-12

陈大发

1

我觉得首先的是要职责单一，不要把过多的东西塞到一个类或者方法中，同一个功能的最好放到一起，或者有联系的地方。尽量把变化的抽离出来，不变的封装起来。

2019-01-07

作者回复

赞，能坚守职责单一，就成功了一半。 我们后面的文章还有专门的讨论。

2019-01-08

hadronw

1

以前看到别人写简洁的代码时，追求简洁时，哇塞，好牛逼！现在明白了，那是傻逼。

0 和 1 最简单了，但是组合在一起后却没什么人看的懂！

最好有些练习、案例之类的

2019-01-05

作者回复

我们先把概念交代好，练手题随后就到。

2019-01-06

So Leung

1

好的代码是"经济的代码",需要衡量编写时间和运行的空间,根据不同的情况综合考虑,在这之外的最重要的是安全!!!!安全是重中之重.

2019-01-04

作者回复

赞这个总结！

2019-01-04

王小勃

1

打卡：
经济--这个词概括好，好的代码并不是绝对的，而是相对的；相对于所处的环境、所服务的规模，根据环境、规模、目标来适配代码做的经济性更好才是最适合的

2019-01-04

作者回复

相对的，你总结用的这个词也很妙。

2019-01-04

cocoa

1

抛开业务逻辑逻辑不说，现在很多IDE都有基本的代码规范插件，比如Java还可使用阿里巴巴规范的插件，如果把这些问题都清理一下的话，质量会有一个基本的保证，也比较经济，不用投入人力去找错误。另外畅想一下，现在机器学习这么流行，可否给自己的业务代码指纹画像，让机器提取优秀业务代码的特征，做出提示。总之要经济还是想办法用机器来解决。

2019-01-04

作者回复

哦，阿里巴巴已经出插件了？机器能干的事，就让它来做。

2019-01-04

阿J

1

写评论也写出bug了，看来真的要好好磨练

2019-01-04

作者回复

□哪有不出bug的人啊

2019-01-04

阿J

1

有时候觉得编码过程眼高手低，有时候觉得手高眼低。最终发现缺乏的原来是缺乏了反复的琢磨的过程。谢谢老师的第一课，会持续跟进学习。

2019-01-04

作者回复

□，我们一起来琢磨

2019-01-04

袁理

0

看了三目运算符的弊端，容易出错，不易阅读，我就找到自己的代码给改成if else了。还发现自己的注释大多是写在语句后面的□，改成写到语句前面了，写注释是为了别人不理解代码也能知道做的什么事情，注释优先

2019-03-15

作者回复

赞！

2019-03-15

Neal

0

对于我来说，好的代码也有底线，统一风格，干净整洁，分目录，尽量职责单一，有一些简单的注释，最怕看到几千行的大方法。和这些胶水程序员一起工作真的每天都心累。

2019-03-07

作者回复

赞，这些问题后面我们还会讨论。

2019-03-08