15 | 编写规范代码的检查清单

2019-02-06 范学雷 来自北京

《代码精进之路》

通过前面十几讲的学习，我们已经把代码“规范”篇的内容学习完了。今天，我们一起把前面讨论到的观点总结一下，并探索一下编写规范代码时的最佳实践检查清单。一份有效的检查清单，可以帮助我们记忆、遵循和执行代码的一系列规范。

标准与指南

在讨论编码规范之前，我们首先要认识什么是标准，什么是指南，以及它们各自的局限性。这样我们就能知道，什么可以做，什么不可以做。

标准是既定的做事方式，它定义了一个事物如何始终如一地达到同样水准的细节。标准的特征在于，它既是必需的，也是强制的。

然而，在现实实践中，能够标准化的东西是有限的。所以就有了指南，它是标准的补充。

指南告诉我们应该采取的总体方向和最佳实践，它是一个指导原则，是建议，不强制执行。用户可以根据具体情况，决定是否遵循相关的条款。

我们所说的编码规范，实际上通常是指导原则。虽然不具备强制性，但我们也有必要区分对不同建议条款的态度，比如对使用“强烈推荐”这样字眼的建议来说，我们就应该格外重视。这样，可以避免不必要的争执，降低复杂性。

为什么需要编码规范?

1. 提高编码的效率

在不损害代码质量的前提下，效率可以节省我们的时间和成本。这种节省不仅仅停留在编码阶段，更体现在整个软件的生命周期里。我在第四篇已经详细解释了这一点。

2. 提高编码的质量

代码的质量在于它和预期规范的一致性。一致、简单、规范的代码易于测试。相反，复杂的代码会加大测试的难度，难以达到合适的测试覆盖率。另外，代码的复用也意味着质量的复用，所以代码的问题会随着它的复用成倍地叠加，提高了软件的使用或者维护成本。

3. 降低维护的成本

代码的维护要求代码必须能够修改，增加新功能，修复已知的问题。如果代码结构的清晰、易于阅读理解，那么问题就容易排查和定位。

4. 扩大代码的影响

要想让更多的人参与，就需要一致的编码风格，恰当地使用文档。要方便他们阅读，便于解释。

使用编码规范可以让一个程序员减少出错，避免不必要的困扰。另外，编写规范的代码也是一种专业、成熟的体现。

编码规范的心理因素

编码风格最原始的依赖因素是人的行为习惯和心理学基础。通过了解一些基本的心理学原理，我们可以更好地理解编码风格的的基本规则。

两种思维模式

我们有两种思维模式，自主模式（快系统）和控制模式（慢系统）。自主模式的运行是无意识的、快速的、不怎么耗费脑力；控制模式需要集中注意力，耗费脑力，判断缓慢，如果注意力分散，思考就会中断。

自主模式在熟悉的环境中是精确的，所作出的短期预测是准确的，遇到挑战时，会第一时间做出反应。然而，它存在成见，容易把复杂问题简单化，在很多特定的情况下，容易犯系统性的错误。比如说，第一印象、以貌取人，就是自主模式遗留的问题。

当自主模式遇到麻烦时，控制模式就会出面解决，控制模式能够解决更复杂的问题。但刻意掌控会损耗大脑能量，而且很辛苦。处于控制模式中太长时间，人会很疲惫，丧失一部分动力，也就不愿意去做启动控制模式了。比如，很多人害怕数学，多是因为控制模式确实很吃力。

自主模式和控制模式的分工合作是高效的，损耗最小，效果最好。快速的、习惯性的决断交给勤快省力的自主模式，复杂的、意外的决断由耗时耗力的控制模式接管。

编码规范中很大一部分内容，是增加共识、减少意外，扩大自主思维模式覆盖的范围，减少控制模式必须参与的内容。熟练掌握编码规范可以逐渐让这一套规则存在于人们的下意识中，这样编码的时候就像我们使用筷子一样，简单又自然。

识别模式

我们能够在这个世界上存活下来，依靠的不是识别完整的情景，也不是从头开始分析每一个情景，而是通过既定的可识别模式，来匹配这个世界。正常期望以外的模式，常常会让我们感到吃惊和困惑，甚至是生气。

模式识别的认知是一柄双刃剑，既是福音也是祸害。它可以帮助我们毫不费力地使用经验，但习惯一旦养成就很难改变，我们不愿意打破旧模式，去学习新模式和接受新技术。

程序员很容易理解和自己编码风格类似的代码。如果编码风格和自己的大相径庭，就会感到焦躁和烦恼。编码风格一旦形成，就难以更改，转变很痛苦。幸运的是，一旦努力转换成新习惯，我们就会喜欢上新的编码风格。

一份好的编码规范，刚开始接受会有些困难。我们甚至会找很多借口去拒绝。但是，一旦接受下来，我们就成功地改进了我们的识别模式。

猜测模式

对于既定模式的识别，是通过猜测进行的。对于每一个新场景，大脑立即会把它起始部分当作一个线索，然后通过拟合所有已知模式的起始部分，来预测模式的其余部分，猜测“言外之意”。我们掌握的信息越少，就越有可能犯错误。比如，在医院看到穿白大褂的，我们默认他们是医护人员。但医护人员的判断标准并不是白大褂。

所以在编写代码时，我们要有意识地提供足够的线索和背景，使用清晰的结构，加快模式的识别，避免造成模式匹配过程中的模糊和混淆带来的理解障碍。

记忆模式

我们的记忆模式有四种，包括感官、短期、工作和长期记忆。

感官记忆是对我们感官体验的记忆，非常短暂（大约三秒钟），比如我们刚刚看到的和听到的。

短期记忆是我们可以回忆的，刚刚接触到的信息的短暂记忆。短期记忆很快，但是很不稳定，并且容量有限。如果中途分心，即便只是片刻，我们也容易忘记短期记忆的内容。

工作记忆是我们在处理认知任务时，对信息进行短暂存贮并且执行操作的记忆。工作记忆将短期记忆和长期记忆结合起来，处理想法和计划，帮助我们做出决策。

长期记忆涵盖的记忆范围从几天到几十年不等。为了成功学习，信息必须从感官或短期记忆转移到长期记忆中。和短期记忆相比，长期记忆记忆缓慢，但是保持长久，并且具有近乎无限的容量。

我们在组织代码时，不要让短期记忆超载，要使用短小的信息快，方便阅读；要适当分割需要长期记忆和短期记忆的内容，比如接口规范和代码实现，帮助读者在工作记忆和长期记忆中组织和归档信息。

眼睛的运动

当我们看一样东西的时候，我们不是一下子就能看清它的全貌。事实上，我们的眼睛一次只能专注于一个很小的区域，忽视该区域以外的内容。当然，我们可以意识到还有更大的区域，然后快速跳转到其他区域。

有时候，我们需要反复研读一段代码。如果这段代码可以在一个页面显示，我们的眼睛就很容易反复移动，寻找需要聚焦的目标。如果这段代码跨几个页面，阅读分析就要费力得多。

当我们阅读时，我们的眼睛习惯从左到右，从上到下移动，所以靠左的信息更容易被接受，而靠右的信息更容易被忽略。

但是，当我们快速阅读或者浏览特定内容时（比如搜索特定变量），眼睛就会只喜欢上下移动，迅速跳过。聚焦区域小，眼睛倾向于上下移动，这就是报纸版面使用窄的版面分割，而不是整幅页面的原因之一。

在编码排版时，要清晰分块，保持布局明朗，限制每行的长度，这样可以方便眼睛的聚焦和浏览。

编码规范的检查清单

下面的这个清单，是我看代码的时候，通常会使用的检查点。如果有检查点没有通过，阅读代码的时候，就要格外留意；编写代码的时候，还要想想有没有改进空间；评审代码的时候，要问清楚为什么这么做，给出改进的建议。

你也可以参考一下。

代码是按照编码指南编写的吗?

代码能够按照预期工作吗?

文件是不是在合适的位置?

支撑文档是不是充分?

代码是不是易于阅读、易于理解?

代码是不是易于测试和调试?

有没有充分的测试，覆盖关键的逻辑和负面清单?

名字是否遵守命名规范?

名字是不是拼写正确、简单易懂?

名字是不是有准确的意义?

代码的分块是否恰当?

代码的缩进是否清晰、整洁?

有没有代码超出了每行字数的限制?

代码的换行有没有引起混淆?

每一行代码是不是只有一个行为?

变量的声明是不是容易检索和识别?

变量的初始化有没有遗漏?

括号的使用是不是一致、清晰?

源代码的组织结构是不是一致?

版权信息的日期有没有变更成最近修改日期?

限定词的使用是不是遵循既定的顺序?

有没有注释掉的代码?

有没有执行不到的代码?

有没有可以复用的冗余代码?

复杂的表达式能不能拆解成简单的代码块?

代码有没有充分的注释?

注释是不是准确、必要、清晰?

不同类型的注释内容，注释的风格是不是统一?

有没有使用废弃的接口?

能不能替换掉废弃的接口?

不再推荐使用的接口，是否可以尽早废弃?

继承的方法，有没有使用 Override 注解?

有没有使用异常机制处理正常的业务逻辑?

异常类的使用是不是准确?

异常的描述是不是清晰?

是不是需要转换异常的场景?

转换异常场景，是不是需要保留原异常信息?

有没有不应该被吞噬的异常?

外部接口和内部实现有没有区分隔离?

接口规范描述是不是准确、清晰?

接口规范有没有描述返回值?

接口规范有没有描述运行时异常?

接口规范有没有描述检查型异常?

接口规范有没有描述指定参数范围?

接口规范有没有描述边界条件?

接口规范有没有描述极端状况?

接口规范的起草或者变更有没有通过审阅?

接口规范需不需要标明起始版本号?

产品设计是不是方便用户使用?

用户指南能不能快速上手?

用户指南的示例是不是可操作?

用户指南和软件代码是不是保持一致?

小结

虽然说编码规范不是强制性的标准，但是如果你能尽可能遵守相同的规范，就会让工作更简单、更高效。

需要特别强调的是，认为写好代码只有一种方法是愚蠢的。虽然编码规范的指导原则是通用的，但是其中的具体细节则依赖于具体的环境，因具体的需求而变化。所以，除了遵循编码规范外，你还要做好随时重审、调整编码规范的准备，保持编码规范的活力，跟得上实际情况的变化。

希望你根据自己的实际情况，不断完善、丰富上面的清单，使这份清单更直观、更容易遵循，保持长久的活力，让它更适合你自己。

一起来动手

正像我们讨论到的，不同的场景，检查清单也不一定相同。如果让你列一个你自己实际工作中需要的编码规范检查清单，会是什么样子的？ 你可以在我上面的清单上加减检查点，或者新做一个列表。欢迎在留言区公布你的检查清单，我们一起来讨论、学习。

另外，推荐一本书《清单革命》。清单能够起到的作用，常常被忽视。这本书告诉我们清单这个小东西，能给我们的工作带来多么巨大的帮助。

如果你觉得这篇文章有所帮助，欢迎点击“请朋友读”，把它分享给你的朋友或者同事。

精选留言 (7)

Sisyphus235
2019-05-22

我相信好的程序开发者实际的代码实践不会占到开发比例的一半，真正的功夫是在代码设计，开发点整理，团队沟通，文档梳理中。

设计清晰结构合理的工作开发效率自然高，定好接口各自开发要比先做着到时候再对接效率高。

代码实践时间占比越高说明前期工作越欠缺，没有准备的仗不好打

作者回复: 同意。历练时间久了，还可能有一个有悖常理的现象：写代码的占用时间越少，可能写的代码还越多。

7

Joey
2020-01-06

请教老师：代码质量量化的实践，可否分享一些，谢谢。

作者回复: 这是一个很难的地方。OpenJDK的做法是依靠大量的测试保证功能的正确性，Code Review保证代码规范性和安全性，Feature Review保证需求和设计的合理性。

2

张亚运
2019-12-13

厉害了，横跨心理学，人体构造学，生物学多个角度说明编码规范的重要性

2

熊猫
2019-05-27

老师能提供下请求https接口的规范代码吗？如常用的okhttp,httpclient等常用框架，十分感谢！

作者回复: 不懂okhttp。

对于httpclient，https和http使用相同的接口，，区别在于HttpRequest.url()是使用HTTPS还是HTTP的地址【1】。HttpClient.Builder构建还也以可以使用SSLContext和SSLParameters配置HTTPS参数【2】。

【1】的例子，把URL里的http前缀改成https就可以了。配置SSLContext和SSLParameters，请参考JSSE Reference Guides【3】。

【1】：https://openjdk.java.net/groups/net/httpclient/intro.html
【2】：https://docs.oracle.com/en/java/javase/11/docs/api/java.net.http/java/net/http/HttpClient.Builder.html
【3】： https://docs.oracle.com/en/java/javase/12/security/java-secure-socket-extension-jsse-reference-guide.html

1

ifelse
2022-07-20

编码规范中很大一部分内容，是增加共识、减少意外，扩大自主思维模式覆盖的范围，减少控制模式必须参与的内容。--记下来

ifelse
2022-07-20

这个清单对我来说太长啦

进化菌
2021-11-23

清单是个好东西，让人有条不紊的做事，而不至于遗漏。
代码规范的检查，一般是在review的时候会完整一点，平时简单看看，所以个人觉得检查清单应该也需要区分快与慢~

作者回复: 快与慢，该怎么理解?

共 2 条评论>