

## 04 | 代码规范的价值：复盘苹果公司的GoToFail漏洞

2019-01-11 范学雷 来自北京

《代码精进之路》



我们在上一讲中讨论了一个优秀的程序员都需要具备哪些良好的品质，第一点就是要熟练掌握一门编程语言。

作为每天都要和代码打交道的人，光是熟练掌握还不够。我们需要像文字写作者一样，对代码有一种“洁癖”，那就是强调代码的规范化。

### 什么是编码规范？

要回答为什么需要编码规范，我们首先要了解编码规范指的是什么。

编码规范指的是针对特定编程语言约定的一系列规则，通常包括文件组织、缩进、注释、声明、语句、空格、命名约定、编程实践、编程原则和最佳实践等。

一般而言，一份高质量的编码规范，是严格的、清晰的、简单的，也是权威的。但是我们有时候并不想从内心信服，更别提自觉遵守了。你可能想问，遵循这样的约定到底有什么用呢？


编码规范可以帮我们选择编码风格、确定编码方法，以便更好地进行编码实践。简单地说，**一旦学会了编码规范，并且严格地遵守它们，可以让我们的工作更简单，更轻松，少犯错误。**

这个问题弄明白了，我们就能愉快地遵守这些约定，改进我们的编程方式了。

## 规范的代码，可以降低代码出错的几率


**复杂是代码质量的敌人。**越复杂的代码，越容易出现問題，并且由于复杂性，我们很难发现这些隐藏的问题。

我们在前面已经讨论过苹果公司的安全漏洞（GoToFail 漏洞），接下来再来看看这个 bug 的伪代码。这个代码很简单，就是两个 if 条件语句，如果判断没问题，就执行相关的操作。

 复制代码

```
1      if ((error = doSomething()) != 0)
2          goto fail;
3          goto fail;
4      if ((error= doMore()) != 0)
5          goto fail;
6 fail:
7      return error;
```

这段代码没有正确地使用缩进和大括号，不是一段符合编码规范的源代码。如果我们使用符合规范的编码方式，这个安全漏洞就自然消失了。你可以看到，下面的代码里，我给 if 语句中加了大括号，代码看起来一下子就简单很多了。

 复制代码

```
1      if ((error = doSomething()) != 0) {
2          goto fail;
3          goto fail;
4      }
5      if ((error= doMore()) != 0) {
6          goto fail;
```

```
7     }  
8   fail:  
9       return error;
```

所以在编码的时候，我们应该尽量使代码风格直观、逻辑简单、表述直接。如果遵守编码规范，我们就可以更清楚、直接地表述代码逻辑。

## 规范的代码，可以提高编码的效率

还记得我们在前面讨论过代码“出道”的重重关卡吗？这些关卡，构成了代码制造的流水线。优秀的代码，来源于优秀的流水线。

如果我们都遵守相同的编码规范，在每一道关卡上，会产生什么样的质变呢？

在程序员编写代码这道关，如果我们规范使用缩进、命名、写注释，可以节省我们大量的时间。比如，如果使用规范的命名，那么看到名字我们就能知道它是一个变量，还是一个常量；是一个方法，还是一个类。

在编译器这道关，我们可以避免额外的警告检查，从而节省时间。还记得我们前面讨论过的 GCC 关于正确使用缩进的编译警告吗？如果有编译警告出现，我们一般都要非常慎重地检查核对该警告有没有潜在威胁。这对我们的精力和时间，其实是不必要的浪费。

还记得 GCC 由于老旧的编程风格的原因，不支持无法访问代码编译错误吗？过度自由的编码风格，有时候甚至会阻碍编译器开发一些非常有用的特性，使得我们无心的过失行为越积累越不好解决。

在代码评审这道关，如果我们不遵守共同的编码规范，这多多少少会影响评审者阅读代码的效率。为什么呢？因为评审者和编码者往往有着不一样的审美偏好。一条评审意见，可能要花费评审者很长时间来确认、评论。然后，源代码编写者需要分析评审意见，再回到流水线的第一关，更改代码、编译、测试，再次提交评审，等待评审结果。

审美偏好一般都难以协调，由此导致的重复工作让编码的效率变得更低了。

在代码分析这道关，编码规范也是可以执行检查分析的一个重要部分。类似于编译器，如果有警告出现，分析警告对我们的精力是一种不必要的浪费；如果过度自由，同样会阻碍代码分析工具提供更丰富的特性。

只要警报拉响，不管处在哪一个关卡，源代码编写者都需要回到流水线的第一关，重新评估反馈、更改代码、编译代码、提交评审、等待评审结果等等。每一次的返工，都是对时间和精力消耗。

**总结一下，在代码制造的每一道关卡，规范执行得越早，问题解决得越早，整个流水线的效率也就越高。**

前一段时间，阿里巴巴发表了《阿里巴巴 Java 开发手册》。我相信，或许很快，执行阿里巴巴 Java 编码规约检查的工具就会出现，并且成为流水线的一部分。对于违反强制规约的，报以错误；对于违反推荐或者规约参考的，报以警告。这样，流水线才会自动促进程序员的学习和成长，修正不符合规范的编码。

## **规范的代码，降低软件维护成本**

代码经过重重关卡，好不容易“出道”了，这样就结束了吗？

恰恰相反，“出道”之后，才是代码生命周期的真正开始。

如果是开源代码，它会面临更多眼光的挑剔。即使是封闭代码，也有可能接受各种各样的考验。“出道”的代码有它自己的旅程，有时候超越我们的控制和想象。在它的旅程中，会有新的程序员加入进来，观察它，分析它，改造它，甚至毁灭它。软件的维护，是这个旅程中最值得考虑的部分。

有统计数据表明，**在一个软件生命周期里，软件维护阶段花费了大约 80% 的成本。**这个成分，当然包括你我投入到软件维护阶段的时间和精力。

举例来说吧，让我们一起来看看，一个 Java 的代码问题，在 OpenJDK 社区会发生什么呢？

在 Java 的开发过程中，当需要代码变更时，我们需要考虑一个问题：使用这些代码的应用是否可以像以前一样工作？

一旦出现了问题，一般有两种可能：要么是 Java 的代码变更存在兼容性问题，要么存在应用使用 Java 规范不当的问题。这就需要确认问题的根源到底是什么。

由于 OpenJDK 是开源代码，应用程序的开发者往往需要调试、阅读源代码。阅读源代码这件事情，在一定程度上，类似于代码评审的部分工作。如果代码是规范的，那么他们的阅读心情就会好一些，效率也就更高。

如果发现了任何问题，可以提交问题报告。问题报告一般需要明确列出存在的具体问题。对于问题报告，也会有专门的审阅者进行研究分析，这个问题描述是否清晰？它是不是一个真正的问题？由谁解决最合适？

很多情况下，报告的审阅者也需要阅读、调试源代码。良好的编码规范，可以帮助他们快速理解问题，并且找到最合适的处理人员。

如果确定了问题，开发人员或者维护人员会进一步评估、设计潜在的解决方案。如果原代码的作者不能提供任何帮助，比如已经离职，那么他们可以依靠的信息，就只有代码本身了。

你看，这个代码问题修改的过程重包含了很多角色：代码的编写者、代码的使用者、问题的审阅者以及问题的解决者，这些角色一般不是同一个人。在修改代码时，不管我们是其中的哪一个角色，遵守规范的代码都能够节省我们的时间。

**很多软件代码，其生命的旅程超越了它的创造者，超越了团队的界限，超越了组织的界限，甚至会进入我们难以预想的领域。**即使像空格缩进这样的小问题，随着这段代码的扩散，以及接触到这段代码人数的增加，由它造成的效率问题也会对应的扩散、扩大。

而严格遵守共同的编码规范，提高代码的可读性，可以使参与其中的人更容易地理解代码，更快速地理解代码，更快速地解决问题。

**编码规范越使用越高效**

除了上面我们说道的好处，编码规范还有一个特点，就是越使用越高效。

比如我们小时候都背通过乘法口诀，如果我问你，3 乘 3 得几？我相信，你立即就会告诉我，答案是 9。不管这时候你是在开车、还是在走路；是在吃饭，还是在玩游戏。

如果我问你，13 乘以 23，结果是多少？除非你经过非常特殊的训练，你不会立即就有答案，甚至你走路的时候，不停下脚步，就算不出这个结果。

如果我问一个还没学过乘法的小孩子呢？3 乘 3 的算术，对于小孩子，也许是一个不小的难题。

对于背通过乘法口诀的我们来说，3 乘 3 的算术根本就不需要计算，我们的大脑可以快速地、毫不费力地、无意识地处理这样的问题。这种系统是我们思维的快系统。快系统勤快、省力，我们喜欢使用它。

而对于 13 乘以 23 的算术，我们的大脑需要耗费脑力，只有集中注意力，才能运算出来。这种系统是我们思维的慢系统。慢系统懒惰、费劲，我们不愿意使用它。

快系统和慢系统分工协作，快系统搞不定的事情，就需要慢系统接管。快系统处理简单、固定的模式，而慢系统出面解决异常状况和复杂问题。

比如上面苹果公司安全漏洞的那个例子，如果我们像乘法表一样熟练使用编码规范，一旦遇到没有使用大括号的语句，我们立即就会非常警觉。因为，不使用大括号的编码方式不符合我们习以为常的惯例，快系统立即就能判别出异常状况，然后交给慢系统做进一步的思考。如果我们没有养成编码规范的习惯，我们的快系统就会无视这样的状况，错失挽救的机会。

所以，我们要尽早地使用编码规范，尽快地培养对代码风格的敏感度。良好的习惯越早形成，我们的生活越轻松。

## 小结

对于编码规范这件事，我特别想和你分享盐野七生在《罗马人的故事》这套书里的一句话：  
**“一件东西，无论其实用性多强，终究比不上让人心情愉悦更为实用。”**




严格地遵守编码规范，可以使我们的工作更简单，更轻松，更愉快。记住，**优秀的代码不光是给自己看的，也是给别人看的，而且首先是给别人看的。**

你有什么编码规范的故事和大家分享吗？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，看看编码规范能不能让你们之间的合作更轻松愉快。

## 一起来动手

下面的这段代码，我们前面用过一次，我稍微做了点修改。我们这次重点来看编码的规范，有哪些地方你看着不顺眼，你会怎么改进？

 复制代码

```
1 package com.example;
2
3 import java.util.Collections;
4 import java.util.List;
5 import javax.net.ssl.SNIHostName;
6
7 class ServerNameSpec {
8     final List<SNIHostName> serverNames;
9
10    ServerNameSpec(List<SNIHostName> serverNames) {
11        this.serverNames = Collections.unmodifiableList(serverNames);
12    }
13
14    public String toString() {
15        if (serverNames == null || serverNames.isEmpty())
16            return "<no server name indicator specified>";
17
18        StringBuilder builder = new StringBuilder(512);
19        serverNames.stream().map((sn) -> {
20            builder.append(sn.toString());
21            return sn;
22        }).forEachOrdered((_item) -> {
23            builder.append("\n");
24        });
25
26        return builder.toString();
27    }
28 }
```

你也可以把这篇文章分享给你的朋友或者同事，一起来讨论一下这道小小的练习题。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

## 精选留言 (28)



**Carlos** 置顶

2019-01-11

我感觉我就是个有代码洁癖的人，写代码时有自己的原则和规范方法，其中做到易读和结构简洁是首要原则，这里面像命名，缩进，空行这种静态规则，有IDE的帮助还算比较容易做到。对于不少人来说比较难做到的可能是代码逻辑的组织。前几天刚看了个同事写的代码，关于视频课程管理的，表结构设计也是他做的。业务上要求课程需要属于某个等级，等级属于类别，类别属于厂商，为了做到同一个课程向不同客户展示不同的价格，又设计了个课程价格关系模型。这几个子项都有对应的业务模型和业务接口类。问题来了，添加课程时，在课程的某一个controller方法里，分别通过getParameter获取参数，构造数据对象，然后调用业务逻辑接口执行插入，最终完成多个业务模型的操作，有的是一对多再来个循环。一个方法里干这么多事情，我都接受了，居然把获取参数和给对象赋值无规律的穿插进行，偶尔还穿插来个参数校验，并且还在方法内部把获取过来的参数加个中文注释。删除课程数据时，需要把关联子模型数据也删除掉，他把各个子模型的删除方法都写到课程的服务里。另外还存在A引用B，B又引用A，同时A又被其他好几个地方引用，不同引用之间通过不同参数走不同分支，并且有些参数在哪个场景下传过来的，他自己也弄不清了。带来这样问题就是写代码前，没把逻辑脉络梳理清楚，完全跟着感觉走，就一个简单的增删改查能写成这样，真怀疑要是把一个后端数据服务的代码给他写不知道能写成啥样。

作者回复：如果逻辑没有理清楚，就开始写代码，就有可能后面越改越乱。

我想起前面有人说的小黄鸭的办法，把代码逻辑讲给小黄鸭听。能讲出来，逻辑就被梳理了一遍。

我在评审代码的时候，如果看到太复杂的代码，我就问作者问题，问为什么这样，问什么不那样做。几个问题下来，代码就会顺利些。

共 2 条评论 >



5



2019-01-11

读烂代码就想抽人。

类似于new一堆对象。然后在几十行以后才使用 如果命名规范还好，如若不规范就要点回去



看。

还有这样奇怪的代码，写了个循环便利集合，但是循环中只有

```
if (list.get (0) != null) {  
return list.get (0) ;  
}
```

每次读这样的代码。全都要研究一下这个循环是干啥的。

还有这样的，解决异常。try包了一大串代码，找哪一行抛的异常很费劲。

还有实体类型字段，String state="1"；

然后后面备注。1代表啥，2代表啥，3代表啥。

不老老实实写个枚举或者常量嘛？

还有代码里的无端空行。写一行，空一行.....一个半屏幕滚动能看完的代码，硬是看了两个半屏幕。

关键是还没法去说，毕竟同事之间。只是默默祈祷，不要让自己维护他的代码就好。

作者回复: 😊我看这个留言，满是欢乐啊。

为什么没办法说呢？一般这种问题，说了就是帮助他提高啊。当然，我也理解有些公司文化不开放。

我替你想两个办法，一个是琢磨琢磨怎么说他会愉快滴接受，并且感谢你。或者，琢磨琢磨怎么搞个规范，让流程帮助他。他变好了，你也工作轻松，心情愉快，对吧？多赢！这种事情，也能给你带来成就感，不信你就试一试😊。

共 2 条评论 >

👍 13



**Y024** 民

2019-01-13

阿里巴巴 java 开发规范，官方已提供了 idea、eclipse 插件，详情可以访问官方链接：<https://github.com/alibaba/p3c/blob/master/README.md>

此外还有FindBugs、PMDPlugin、CheckStyle  
、JavaNCSS、sonarlint，可以多管齐下，为你保驾护航。

作者回复: 谢谢提供这么多资源🙏

共 2 条评论 >

👍 10



**richy**

2019-01-11

范老师，阿里的代码规约检查插件已经有了，还挺好用的

作者回复: 谢谢你! 我要记得下次把这个修订了。



👍 6

**cocoa**

2019-01-11

- 1.属性明确加上private
- 2.泛型赋值的时候自动判断类型，不用显示加
- 3.if的第一个或条件不需要
- 4.if使用大括号
- 5.map一般用来转换，这块用迭代foreach就行了
- 6.不用使用顺序迭代，本来集合list就有顺序
- 7.toString显示标明override

作者回复: > 1.属性明确加上private

如果是个公开类，需要使用private修饰符。如果是内部类，使用缺省的修饰符也可以。

> 2.泛型赋值的时候自动判断类型，不用显示加  
是的。

> 3.if的第一个或条件不需要  
如果把缺省的无参构造函数补上，这个条件就明显地不需要了。

> 4.if使用大括号  
对的。

> 5.map一般用来转换，这块用迭代foreach就行了

> 6.不用使用顺序迭代，本来集合list就有顺序  
是的，foreach足以。

> 7.toString显示标明override  
是的，这是一个违反规范的地方。



👍 6



阔别

2021-05-25

我不明白换行这个事情有什么必要来回说的, 现在工具不都可以格式化吗, 我无意diss作者, 但是我已经看到第四节了, 依然没有看到真正有任何价值的东西



👍 2



2019-01-14

“我替你想两个办法，一个是琢磨琢磨怎么说他会愉快滴接受，并且感谢你。或者，琢磨琢磨怎么搞个规范，让流程帮助他。他变好了，你也工作轻松，心情愉快，对吧？多赢！这种事情，也能给你带来成就感，不信你就试一试😊。”

为了看到规范代码。我做了以下的事：

1. 自己先更加严格要求自己的代码。
2. 参加了阿里巴巴代码规范的认证考试，并拿到了证书。
3. 把证书给领导看一眼（证明我自己是规范的），然后说规范很有用。
4. 领导表示，嗯，他也觉得是这样。
5. 领导在群里通知，建议大家考一下，费用公司报销。
5. 几个月过去了，我还是唯一一个我们单位执证上岗的。
6. 报了范老师的课程。无法要求别人，但是更不能停止精进自己！最起码不能让别人骂我的代码！

希望这段留言不会被我同事或者领导看到，紧张。

不过，被看到了，貌似也有好处，没准我们公司的代码明天就规范化了呢。

作者回复：赞！领导和同事看到你的留言也没问题的，规范是多赢的局面，大家都进步。规范这东西，用的人越多力量越大，效率越高。

同事们没有规范，大概率是还没有意识到规范的好处。前几篇文章我自己都觉得都有点啰嗦。但是我就是要把这些基本的理念，翻过来调过去的讲，从不同的角度讲。有了这些意识以后，技术的东西都是小事了。我也建议你分享一下专栏的这几篇文章给同事们。

我想一想有什么形式，可以让三五个小伙伴组成一个小组，把代码规范就启动起来，还不增加大家的工作负担。也许后续我们可以写一个专题文章。专栏的小伙伴们，也帮着想想办法。



👍 2



槛外人

2019-01-14

希望能多点最佳实践，最近几期好像有点重复了

作者回复: 好的, 以后多考虑加点实践。



👍 2



**Linuxer**

2019-01-12

看自己的代码不顺眼, 看别人的也是, 开源代码很多就看着赏心悦目。总感觉那方面还需要加强, 总是不得其法

作者回复: 我们这个专栏就是讨论怎么写好代码的。我们慢慢来。 讨论区有很多高手, 多参与讨论, 多做练手题。一定会有收获的。

共 2 条评论 >

👍 2



**秦凯**

2019-01-11

分享两条简单、有价值的实践:

在编辑器中设置使用指定的空格数 (一般四个) 替换tab键;

范老师之前提到的, 用if else 替换三目运算符, 以及用 `i += 1;` 替换 `++i;`等, 可以让代码逻辑更易于理解, 降低阅读成本。

作者回复: 好使 😊, tab键, 可以在IDE设置好, vi也可以设置, 把tab换成四个空格。



👍 2



**ownraul**

2019-01-18

代码首先是给自己看的, 然后是给别人看的, 最终还要给以后的自己看的  
只有让人一眼望去, 其中想要表达的逻辑能清晰明了, 一览无遗, 这才是好代码



👍 1



**Numbpad1**

2022-11-17 来自广东

```
import javax.net.ssl.SNIHostName;  
import java.util.Collections;  
import java.util.List;
```

```
public class ServerNameSpec {  
    final List<SNIServerName> serverNames;  
  
    ServerNameSpec(List<SNIServerName> serverNames) {  
        this.serverNames = Collections.unmodifiableList(serverNames);  
    }  
  
    @Override  
    public String toString() {  
        if (serverNames == null || serverNames.isEmpty()) {  
            return "";  
        }  
        StringBuilder builder = new StringBuilder(512);  
        serverNames.forEach((_item) -> {  
            builder.append(_item.toString());  
            builder.append("\n");  
        });  
        return builder.toString();  
    }  
}
```



**ifelse**

2022-07-15

优秀的代码不光是给自己看的，也是给别人看的，而且首先是给别人看的。--记下来



**蝴蝶**

2022-05-13

Stream的缩进烧脑 override加上 成员变量私有 我想改的



**进化菌**

2021-11-12

规范规范，有规矩有范例，遵守规矩效仿案例。

练习里，有看到没有使用合理的缩进和大括号.....其实，最近看自己一个月前的代码，一个

方法了放过重的逻辑，看的也是很头大。

代码如写诗，总得努力改变，让Ta如同自己一般清晰稳健的劳动~



**玄兴梦影**

2020-03-26

代码如诗！



**丁丁历险记**

2020-01-14

对o 为啥不加大括号??? 脑子进水了???



**華**

2019-08-27

编码规范除了个人要有良好的编码习惯外，最好还是能有一些工具来辅助，比如阿里的编码规范插件，还有团队协作编码，基础的规则，可以使用EditorConfig制定。



**Sisyphus235**

2019-05-21

代码不是写给自己看的，也没有代码不需要迭代开发，规范的代码能提高协作的效率



**小田**

2019-01-12

代码规范是协作的基础，质量的保证，因此促成了开发的高质量、高效率

