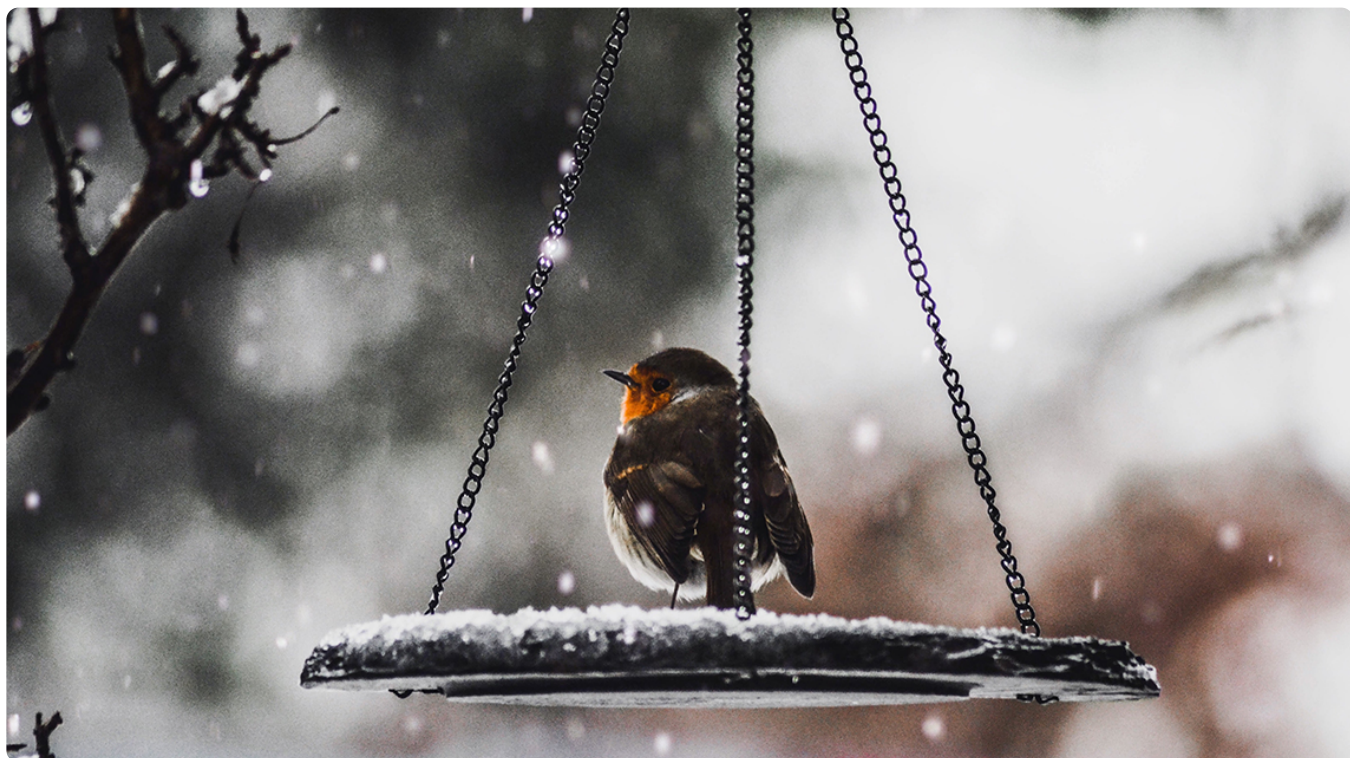


## 02 | 把错误关在笼子里的五道关卡

2019-01-07 范学雷



讲述：黄洲君

时长 12:37 大小 17.35M



上一讲中，我们一起讨论了什么是优秀的代码。简而言之，优秀的代码是经济、规范、安全的代码。在平时的工作中，我们要朝着这个方向努力，时常站在团队、流程、个人能力的角度去思考优秀代码。

作为一名软件工程师，我们都想写出优秀的代码。可是，怎么才能编写出经济、规范、安全的代码呢？这是个大话题，相信你之前也有过思考。

### 无心的过失

开始之前，我先给你讲个曾经发生过的真实案例。2014 年 2 月，苹果公司的 iOS 和 OS X 操作系统爆出严重的安全漏洞，聪明的黑客们可以利用这一漏洞，伪装成可信网站或者服务，来拦截用户数据。而造成这一漏洞的原因，也让业界专家大跌眼镜。

下面我用 **C 语言的伪代码**来给你简单描述下当时的漏洞情况。

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

其实这段代码非常简单，它有两个判断语句，如果判断条件成立，那就执行 “goto fail” 语句，如果不成立，那就跳过判断语句继续执行。上面的 “goto fail” 语句，它的意思是略过它之后的所有语句，直接跳转到标有 “fail” 语句的地方，也就是第 6 行。

我们来分析下，第一个判断条件（第一行和第二行），如果 error 不等于零，那就跳转到 fail 语句，这逻辑上没什么问题。而第三行，没有任何附加条件，就能直接跳转到 fail 语句，也就是说，它下面的代码永远也无法执行，这里是不是有问题？是的，漏洞就是出在这里。

这一行多余的代码就是导致苹果操作系统那个安全漏洞的罪魁祸首。2014 年 2 月 21 日，苹果发布了相关的安全补丁，你随便一搜 “GoTo Fail 漏洞” 就能找到相关的细节，我这里不赘述了。

我们每天仰慕的苹果操作系统出现这样 “低级” 的错误，你是不是有点惊讶？这么一个 “简单” 的错误，引发了一个非常严重的安全漏洞，是不是也有点出乎意料？上面的错误，简单看，就是复制的时候多复制了一行，或者因为时间关系，或者因为粗心大意，苹果的工程师硬是没检查出来。这在我们平时的工作中，也经常出现。

这个具有重大杀伤力的 bug 是如此的 “幼稚”，如此的 “好玩”，如此的 “萌萌哒”，以至于到现在，人们还可以买到印有 “GoTo Fail” 的 T 恤衫，更别提业界对于这个问题的兴趣了。有很多文章，专门研究这一个 “低级” 安全漏洞；甚至有人探讨这个 “低级” 错误对于计算机软件教育的积极影响。

所有的危机都不应该被浪费，这一次也不例外。这些年，我也一直在思考**为什么我们会犯如此 “低级” 的错误**？即使是在苹果这样的大公司。反过来再想，我们应该如何尽可能避免类似的错误呢？

## 人人都会犯错误

没有人是完美的，人人都会犯错误。这应该是一个共识。这里面既有技术层面的因素，也有人类的行为模式的因素，也有现实环境的影响。我们在此不讨论人类进化和心智模式这样的严肃研究成果。但是，有两三个有意思的话题，我想和你聊聊。

第一个比较普遍的观点是好的程序员不会写坏的代码，要不然，就是他还不够优秀。我尊重这个观点背后代表的美好愿望，但是这个观点本身我很难认同。它一定程度上忽视了人类犯错误的复杂性，和影响因素的多样性。

我认为，即使一个非常优秀的程序员，他主观上非常认真，能力又非常强，但他也会犯非常“低级”、“幼稚”的错误。所以，你不能因为苹果那个程序员，犯了那个非常低级的错误，就一棒子把他“打死”，认为他不是一个好的程序员。

第二个更加普遍的观点是同样的错误不能犯第二次。作为一名程序员，我同样尊重这个观点背后代表的美好期望。但是，我想给这个观点加一点点限制。这个观点应该是我们对自身的期望和要求；对于他人，我们可以更宽容；**对于一个团队，我们首先要思考如何提供一种机制，以减少此类错误的发生**。如果强制要求他人错不过三，现实中，我们虽然发泄了怨气，但是往往错失了工作机制提升的机会。

第三个深入人心的观点是一个人犯了错误并不可怕，怕的是不承认错误。同样的，我理解这个观点背后代表的美好诉求。这是一个深入人心的观点，具有深厚的群众基础，我万万不敢造次。在软件工程领域，我想，在犯错这件事情上，我们还是要再多一点对自己的谅解，以及对他人的宽容。错误并不可怕，你不必为此深深自责，更不应该责备他人。要不然，**一旦陷入自责和指责的漩涡，很多有建设意义的事情，我们可能没有意识去做；或者即使意识到了，也没法做，做不好**。

我这么说，你是不是开始有疑惑了：人人都会犯错误，还重复犯，还不能批评，这怎么能编写出优秀的代码呢？换句话说就是，**我们怎么样才会少犯错误呢？**

## 把错误关在笼子里

人人都会犯错误，苹果的工程师也不例外。所以，“GoTo Fail”的“幼稚”漏洞，实在是情理之中。可是，这样的漏洞是如何逃脱重重“监管”，出现在最终的发布产品中，这多多少少让我有点出乎意料。

我们先来看看，这个错误是经过了怎样的“工序”，穿越了多少障碍，需要多少运气，最终才能被“发布”出来。

我把这样的工序总结为“五道关卡”。

## 第一道关：程序员


**提高程序员的修养，是一个永不过时的课题。从别人的失败和自己的失败中学习、积累、提高，是一个程序员成长的必修课。**我知道，这是你和我一直都在努力做的事情。

第三行的“GoTo Fail”，实在算得上“漏网之鱼”，才可以逃过哪怕最平凡的程序员的眼睛，堂而皇之地占据了宝贵的一行代码，并且狠狠地玩耍了一把。

现在我们可以再回过头来看看那段错误代码，如果是你写，你会怎么写呢？从你的角度来看，又有哪些细节可以帮助你避免类似的错误呢？这两个问题，你可以先停下来 1 分钟，想一想。

在我看来，上面那段代码，起码有两个地方可以优化。如果那位程序员能够按照规范的方式写代码，那“GoTo Fail”的漏洞应该是很容易被发现。我们在遇到问题的时候，也应该尽量朝着规范以及可持续改进的角度去思考错误背后的原因，而非一味地自责。

首先，**他应该正确使用缩进**。你现在可以再看下我优化后的代码，是不是第三行的代码特别刺眼，是不是更容易被“逮住”？

 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;
```

其次，**他应该使用大括号**。使用大括号后，这个问题是不是就自动消失了？虽然，多余的这一行依然是多余的，但已经是没有多大危害的一行代码了。



```
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;
```

从上面这个例子里，不知道你有没有体会到，好的代码风格带来的好处呢？工作中，像苹果公司的那位程序员一样的错误，你应该没少遇到吧？那现在，你是不是可以思考如何从代码风格的角度来避免类似的错误呢？

魔鬼藏于细节。很多时候，**优秀的代码源于我们对细节的热情和执着**。可能，你遇到的或者想到的问题，不是每一个都有完美的答案或者解决办法。但是，**如果你能够找到哪怕仅仅是一个小问题的一个小小的改进办法，都有可能给你的代码质量带来巨大的提升和改变**。

当然，你可能还会说，我代码风格不错，但是那个问题就是没看到，这也是极有可能的事情。是这样，所以也就有了第二道工序：编译器。

## 第二道关：编译器

编译器在代码质量方面，作为机器，恪尽职守，它可以帮助我们清除很多错误。还是上面的漏洞代码为例子，这一次其实编译器的防守并没有做好，因为它毫无察觉地漏过了多余的“GoTo Fail”。

在 Java 语言里，对于无法访问的代码（第三行后的代码），Java 编译器就会及时报告错误。而在 2014 年 2 月的 GCC 编译器里，并没有提供这样的功能。

至今，GCC 社区对于无法访问代码的检查，还没有统一的意见。然而，GCC 社区并没有完全浪费这个“GoTo Fail”的问题。为解决类似问题，从 GCC 6 开始，GCC 社区为正确使用缩进提供了一个警告选项（-Wmisleading-indentation）。如果代码缩进格式没有正确使用，GCC 就会提供编译时警告。现在，如果我们启用并且注意到了 GCC 编译器的警告，犯类似错误的机会应该会大幅度地降低了。

在这里，我要提醒你的是，对于编译器的警告，我们一定要非常警觉。能消除掉所有的警告，你就应该消除掉所有的警告。就算实在没有办法消除掉编译警告，那你也一定要搞清楚警告产生的原因，并确认编译警告不会产生任何后续问题。

### 第三道关：回归测试（Regression Testing）

一般地，软件测试会尽可能地覆盖**关键逻辑和负面清单**，以确保关键功能能够正确执行，关键错误能够有效处理。一般情况下，无论是开发人员，还是测试人员，都要写很多测试代码，来测试软件是否达到预期的要求。

另外，这些测试代码还有一个关键用途就是做回归测试。如果有代码变更，我们可以用回归测试来检查这样的代码变更有没有使代码变得更坏。

上述的“GoTo Fail”这样的代码变更，涉及到一个非常重要的负面检查点。遗憾的是，该检查点并没有包含在回归测试中；或者，在这个变更交付工程中，回归测试并没有被执行。


软件测试没有办法覆盖所有的使用场景。但是，我们千万要覆盖关键逻辑和负面清单。一个没有良好回归测试的软件，很难保证代码变更的质量；也会使得代码变更充满不确定性，从而大幅地提高代码维护的成本。

### 第四道关：代码评审（Code Review）

**代码评审是一个有效的在软件研发过程中抵御人类缺陷的制度。**通过更多的眼睛检查软件代码，被忽视的错误更容易被逮住，更好的设计和实现更容易浮现出来。

那代码评审是怎么实现的呢？一般情况下，代码评审是通过阅读代码变更进行的。而代码变更一般通过某种形式的工具呈现出来。比如 OpenJDK 采用的 [Webrev](#)。你可以访问[我的一个代码评审使用的代码变更页面](#)，感受下这种呈现方式。

回到上面那个“GoTo Fail”的代码变更，看起来什么样子呢？下面是其中的一个代码变更版本示例：

 复制代码

```
1 if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
2     goto fail;
```

```
3 + goto fail;
4 if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
5     goto fail;
```

添加的这行代码，还是相当刺眼的。多一些眼睛盯着这些代码，多一些形式展现这些变更，就会大幅度地降低问题藏匿的几率。

上述的“GoTo Fail”这样的代码变更，怎么就逃过代码评审者的眼睛呢？我想说的是，评审者也是人，我们不能期望评审者能发现所有的问题。

## 第五道关：代码分析（Code Analysis）

静态代码分析（Static Code Analysis）是通过对源代码的检查来发现潜在问题的一种软件质量保障方式。有很多静态代码分析工具可以帮助你检查代码缺陷，比如说商业软件 Coverity，以及开源软件 FindBugs。你可以试试看，有哪些工具可以检测到这个“GoTo Fail”问题。

代码覆盖率（Code Coverage）是一个反映测试覆盖程度的指标。它不仅仅量化测试的指标，也是一个检测代码缺陷的好工具。如果你的代码覆盖率测试实现了行覆盖（Line Coverage），这个“GoTo Fail”问题也很难闯过这一关。

很显然，苹果的这一关也没有拦截住“GoTo Fail”。这样，“GoTo Fail”就像千里走单骑的关云长，闯过了五关（有些软件开发流程，也许会设置更多的关卡）。

## 代码制造的流水线


我们分析了这重重关卡，我特别想传递的一个想法就是，**编写优秀的代码，不能仅仅依靠一个人的战斗**。代码的优秀级别，依赖于每个关卡的优秀级别。高质量的代码，依赖于高质量的流水线。每道关卡都应该给程序员提供积极的反馈。这些反馈，在保障代码质量的同时，也能帮助程序员快速学习和成长。

可是，即使我们设置了重重关卡，“GoTo Fail”依然“过关斩将”，一行代码一路恣意玩耍。这里面有关卡虚设的因素，也有我们粗心大意的因素。我们怎么样才能打造更好的关卡，或者设置更好的笼子？尤其是，**身为程序员，如何守好第一关？**

欢迎你在留言区说说自己的思考。下一讲，我们再接着聊这个话题。

## 一起来动手

下面的这段代码，有很多疏漏的地方。你看看自己读代码能发现多少问题？上面我们讨论的流程能够发现多少问题。不妨把讨论区看作代码评审区，看看在讨论区都有什么不同的发现。

 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 serverNames;
9
10    ServerNameSpec(List serverNames) {
11        this.serverNames = Collections.unmodifiableList(serverNames);
12    }
13
14    public void addServerName(SNIHostName serverName) {
15        serverNames.add(serverName);
16    }
17
18    public String toString() {
19        if (serverNames == null || serverNames.isEmpty())
20            return "<no server name indicator specified>";
21
22        StringBuilder builder = new StringBuilder(512);
23        for (SNIHostName sn : serverNames) {
24            builder.append(sn.toString());
25            builder.append("\n");
26        }
27
28        return builder.toString();
29    }
30 }
```

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。



# 代码精进之路

你写的每一行代码都是你的名片

范学雷

Oracle 首席软件工程师  
Java SE 安全组成员  
OpenJDK 评审成员



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得转载

上一篇 01 | 从条件运算符说起，反思什么是好代码

下一篇 Q&A加餐 | 关于代码质量，你关心的那些事儿

## 精选留言 (43)

写留言



范学雷 置顶  
2019-01-18

7

有一个问题我搞错了，部分留言的回复我已经不能修改了。请见谅！

一个类，如果写了构造方法，不论有没有构造参数，就没有缺省的构造方法了(无参的构造方法)。所以，我们的例子中，只要写了有参的构造方法，并且serverNames没有初始化为空，final的serverNames就不会是空值了。

展开



古城痴人  
2019-01-07

12

1. 第7行：ServerNameSpec建议增加访问修饰符：public

2. 第11行：返回的是 `UnmodifiableList` 类型的List，但是在15行中使用了add方法，会抛：`UnsupportedOperationException`异常；
3. 第20行：没有缩进；也没有使用大扩号来包裹代码块。
4. 第23行：`serverNames`没有使用泛型，所以直接使用`SNIServerName`会编译不过。

展开 ▾

作者回复: 找到了好几个问题，很赞！

> 1. 第7行：`ServerNameSpec`建议增加访问修饰符：`public`

要是公开类，需要加`public`；包内部的类，可以使用缺省的修饰符。

> 2. 第11行：返回的是 `UnmodifiableList` 类型的List，但是在15行中使用了add方法，会抛：`UnsupportedOperationException`异常；

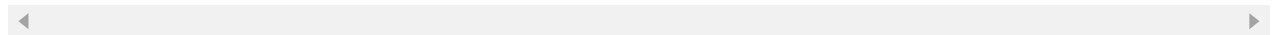
👉，是的。如果肉眼看不到，这是一个测试可以测出的错误。

> 3. 第20行：没有缩进；也没有使用大扩号来包裹代码块。

是的，需要使用大扩号和缩进，两个都要用。

> 4. 第23行：`serverNames`没有使用泛型，所以直接使用`SNIServerName`会编译不过。

是的，这是一个编译器可以发现错误。



王智

2019-01-07

👍 4

1. class使用`public`修饰比较好,一个类有一个主类
2. `final`变量应该初始化
3. `Collections.unmodifiableList()`生成的List无法修改
4. if条件尽量使用括号,下面的return应该缩进
5. List没有指定泛型,遍历就不是`SNIServerName` 类型,应该是Object...

展开 ▾

作者回复: ( 修改后的回复, 我遗漏了已经没有缺省构造函数的问题, 谢谢@kenes孙 )

> 1. class使用`public`修饰比较好,一个类有一个主类

嗯, 要是公开类, 需要使用`public`修饰符。 要是包内部类, 缺省的也可以, 只能在包内使用。

> 2. `final`变量应该初始化

`final`变量在构造函数里初始化也可以。

> 3. `Collections.unmodifiableList()`生成的List无法修改

对的, 这是一个绕弯的问题, 你找到了👉!

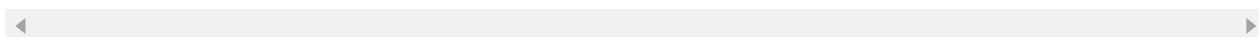
> 4. if条件尽量使用括号,下面的return应该缩进  
是的, 括号和缩进都要有!

> 5. List没有指定泛型,遍历就不是SNIServerName 类型,应该是Object  
对的, 我们再声明时, 就应该把泛型类型这个问题处理好。

> 6 builder.append追加两次可以改成一次,节省运算  
非常好的观点!

> 以上就是我的见解,可能不正确,还请谅解

留言区就是大家畅所欲言的、开放的地方。“人人都会犯错误”那一小节,我就是想让大家彻底放下犯错误的任何包袱。想说什么就说什么, 放开点。



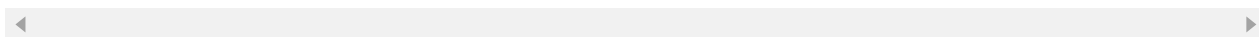
**vector**

2019-01-07

👍 3

评论区高手如云,学到了好多。我想请教下老师,以Spec结尾命名这个类,是有什么说法吗,看到jdk源码也有好多这样的包名和类名,是特定的,指定的意思吗?

作者回复: Spec一般是Specification的缩写,表示这个类表示某种实际的规范,包含特定的参数。比如RSAPublicKeySpec,表示一个RSA公开密钥是有那几个参数组成的。



**Kai**

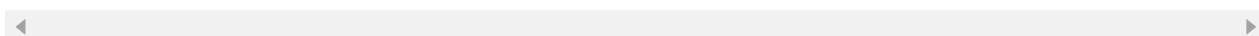
2019-01-07

👍 3

小黄鸭就是把你的代码逻辑解释给一个玩具听

展开 ▾

作者回复: 谢谢了



**hz**

2019-01-07

👍 2

15行和24行可能引发空指针异常

展开 ∨

作者回复: 为什么会引起空指针异常? 能多解释下吗?

◀ ▶



落叶飞逝的...

2019-01-14

👍 1

关于课后思考题,除了其他同学的回答的,我再加一个,就是这段代码没注释。目前可能这段代码比较短,通过代码阅读可以知晓这段代码做什么功能。但如果实际项目的一大段代码没注释,那真的很痛苦!!!

展开 ∨

作者回复: 赞!这真的是一个很大的问题!

◀ ▶



kenes孙

2019-01-13

👍 1

已经有了有参的构造函数,就没有了默认的空参构造函数,所以根本new不了,自然就没有空构造函数调用add的问题了吧.....

作者回复: 你是对的,这一点我遗漏了。谢谢!

我要回头改改给其他人的留言,免得误导了大家。

◀ ▶



夏落若

2019-01-09

👍 1

自己看出的问题加上看留言别人发现的问题,总结一下所有问题如下:

第7行, class使用public修饰

第8行, final变量应该初始化,且不能在构造函数中修改serverNames的引用  
空构造函数,调用add会报错

第11行, Collections.unmodifiableList()生成的List无法修改...

展开 ∨

作者回复: (修改后的回复,我遗漏了已经没有缺省构造函数的问题,谢谢@kenes孙)

总结的好!

> 第7行，class使用public修饰

嗯，如果是公开的类，就要有访问权限。如果是包内部的类，使用缺省的访问权限也可以。

> 第8行，final变量应该初始化，且不能在构造函数中修改serverNames的引用  
final变量，在构造函数中初始化就行。

> 空构造函数，调用add会报错

空构造函数没有声明，使用了带参的构造函数，缺省的空构造函数就没有了。

> 第11行，Collections.unmodifiableList()生成的List无法修改  
这个真不是，在构造函数中，可以初始化final变量。不过，也依赖于你最后怎么改的这个代码。  
你要是第8行初始化了，这里就不能再次初始化了。

> 第15行，List没有指定泛型,遍历就不是SNIServerName 类型,应该是Object  
嗯，我们应该在第8行声明时，就把类型这个问题解决掉。

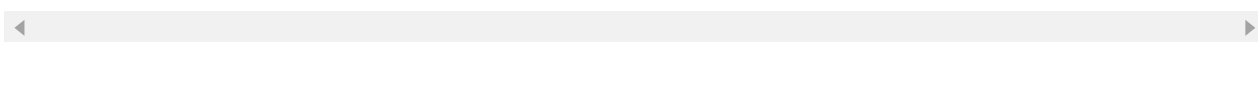
> 第19行，if条件尽量使用括号,下面的return应该缩进  
是的，缩进和括号，都要有！

> 第23行，for循环可能会空引用。循环之前需要判断serverNamers是否是null  
对的，需要检查空值。

> 第24，25行，builder.append追加两次可以改成一次,节省运算  
嗯，可以写成一行；写两行也没什么毛病。

你看你看，你上面的找问题，其实就是代码评审最关键的部分。人人都可以做，人人都可以找出问题。鉴于人人都会犯错误，也不要求每个问题都找对，所有的问题都找到。看代码的眼睛越多，代码的错误隐藏的机会就越小。

如果我们这样看别人的代码，看自己的代码，不用多长时间，代码质量就会有显著的提升，编码越来越轻松。加油！



大於一

2019-01-08

1

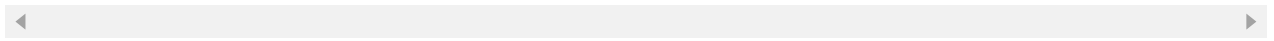
回归测试其实怎么测? 不懂

展开



作者回复: 回归测试, 简单的说, 就是每做一个变更, 把测试都跑一遍, 免得变更引起我们意想不到的麻烦。

找找有没有这方面的专栏。如果没有, 你在给我留言, 我们看看怎么样介绍些这方面的内容。



等

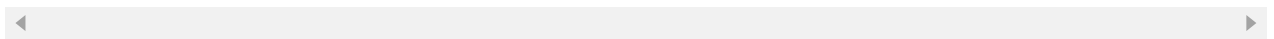
2019-01-08

1

空构造函数, 调用add会报错的吧

展开

作者回复: 是的, 这是一个隐蔽的较深的问题。高手!



sdmanooo

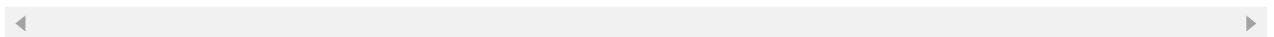
2019-01-07

1

这个bug很可能在准备上线合并代码的时候给合并错了

展开

作者回复: 这也是其中一种猜想。



阿官(^·...

2019-01-07

1

1、成员变量 `serverNames`被`final`修饰, 需要初始化, 且不能在构造函数中修改 `serverNames`的引用

2、`toString()`方法中

2.1 `if`语句需要 `{ }`

2.2 `return`语句没有缩进...

展开

作者回复: ( 修改后的回复, 我遗漏了已经没有缺省构造函数的问题, 谢谢@kenes孙 )

&gt; 1、成员变量 `serverNames`被`final`修饰, 需要初始化, 且不能在构造函数中修改 `serverNames`的引用

`final`变量可以在构造函数里初始化。

&gt; 2、toString()方法中

&gt; 2.1 if语句需要 {}

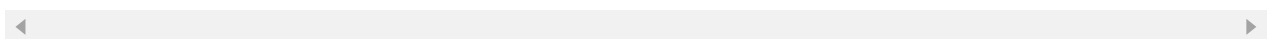
对的，要坚持使用大括号！

&gt; 2.2 return语句没有缩进

是的，缩进没做好，格式很难看。

&gt; 2.3 for循环中的变量sn.toString()可能会出现NullPointerException

这个问题找的好！



Kai

2019-01-07

👍 1

1 ) serverNames 应该申明为 private。

2 ) L19 应该使用大括号。L20应该注意缩进

3 ) builder 有可能会stack overflow

4 ) serverNames并没有申明List<String>，所以sn有可能不是String，也就没有toString()方法

展开 ▾

作者回复:

> 1 ) serverNames 应该申明为 private。

嗯，如果是公开的类，应该申明为 private。

> 2 ) L19 应该使用大括号。L20应该注意缩进

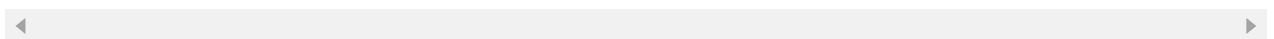
对的，就应该这样。大括号和缩进，一个都不能少。

> 3 ) builder 有可能会stack overflow

能多解释一点，为什么会发生stack overflow吗？

> 4 ) serverNames并没有申明List<String>，所以sn有可能不是String，也就没有toString()方法

Java的每个对象，都有一个toString()方法，继承自Object.toString()。不过，这一行的确有个问题，你再想想看。



玄源

1



2019-01-07



构造方法返回的是不可修改的List，add的时候应该会报错

展开

作者回复: 大大的赞！这是一个绕了个大弯的问题，你看出来了！



chengang

2019-01-07



除了@pyhhou提到命名规范也很重要，servername和servername肉眼很难区分,512定义为常量是否更加合理

作者回复:

> servernames和servername肉眼很难区分

嗯，这个点很好！

> 512定义为常量是否更加合理

这也是一个很棒的观察点，是要考虑这个问题的。



1811876295...

2019-02-12



看了一下SNIHostName 的取值返回在0-255之间，否则会抛异常，512的容量肯定大了，需要设置在0-255之间。

这是我非常非常非常喜欢的一点！这一个问题找的太好了！我们要去看不熟悉的类的规范，一定是看过才会找到这个问题。

...

展开

作者回复: 抱歉，我应该解释下这个背景。

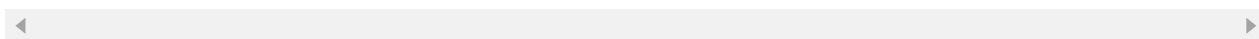
SNIHostName一般用在TLS的连接中，用来指明所连接的服务器的域名。比如，我们使用https://www.example.com/，那么SNIHostName就应该设置成"www.example.com"。建立连接时，这个域名会被校验，已确保的确时连接到了"www.example.com"，而不是一个冒牌的网站。

为了更多的灵活性，TLS的规范定义可以使用多个域名。实际上，一般情况下（比如HTTPS）一个连接仅使用一个域名。ServerNameSpec就是用来描述规范定义的，而SNIHostName就是

用来描述一个域名的。

由于一般情况下，只有一个域名，所以初始化内存够一个域名用的效率就会好一些。这个问题之所以找的好，是因为，如果不去阅读相关的规范，是不会想到这一点的，这是非常专业的内容。

类似的初始化设置，如果初始化的容量和需要的容量一样大小、或者大一点点，都没有问题。但是，如果容量并不显而易见，需要计算，比如要遍历一个大的列表，我们就大致估计一个数值就好了，不要去遍历列表。如果也估计不出来，使用512算是一个常见的选择。

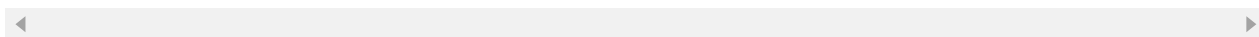


2019-01-28



看到老师的Code Review示例的时候，瞬间感觉高大上。高质量的代码，依赖于高质量的流水线，那么问题来了，如何建立中小企业的高质量的代码流水线呢？请老师指点迷津。

作者回复: 使用好开源工具，小团队也可以做。这个话题，我们找个时间，专门聊一聊。



William

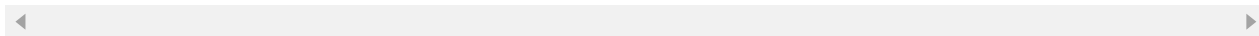
2019-01-28



针对 条件判断语句：go fail 这样的判断之后的代码块，需要写完整的 { }，是蛮重要的。

展开 ∨

作者回复: 必须的，好多规范强制要求使用括号。



茜茜若紫\_...

2019-01-26



我以前一直没有意识到单元测试的重要性，直到最近才感受到人真的不可能完全不犯错误，复制黏贴多一行少一行，拷贝过来的代码是否合用？改了一个逻辑是否会影响很多其他的地方，这些光靠人考虑实在是太难了。但是如果之前写好了单元测试用例只要跑一遍，那代码逻辑是否能正常执行就很清楚了。回归一遍所有的单测，其他地方是否受到影响基本上就能够直接看出来。省了不知多少事。但是首先得要留出一定的时间把单测写...

展开 ∨

作者回复: 稍微大一点的软件，稍微生命周期长一点的代码，单元/回归测试对效率的提高，真不是一点半点的。写一次测试，以后就每次都能帮我们看大门。

