

## 20 | 简单和直观，是永恒的解决方案

2019-02-18 范学雷

代码精进之路

[进入课程 >](#)



讲述：刘飞

时长 09:03 大小 16.57M



上一次，我们聊了影响代码效率的两个最重要的因素，需求膨胀和过度设计。简单地说，就是找到要做的事情，做的事情要少。接下来，我们来聊聊怎么做这些事情。其中，我认为最重要的原则就是选择最简单、最直观的做法。反过来说，就是不要把事情做复杂了。

要想简单直观，我们要解决两个问题。第一个问题是，为什么要简单直观？第二个问题是，该怎么做到简单直观？

### 为什么需要简单直观？

简单直观，看似是一条每个人都能清楚明白的原则。事实上，这是一个非常容易被忽略的原则。如果我们没有对简单直观这个原则有一个基本的认识，就不太可能遵循这样的原则。

我们都喜欢原创和挑战，来展示我们的聪明才智。而简单直观的事情，显而易见的解决办法，似乎不足以展示我们的智慧和解决问题的能力。

遗憾的是，在软件世界里，一旦我们脱离了简单直接的原则，就会陷入行动迟缓、问题倍出的艰难境地。简洁是智慧的灵魂，我们要充分理解这一点。

## **简单直观是快速行动的唯一办法**

我们真正要的不是简单直观的代码，而是轻松快速的行动。编写简单直观的代码只是我们为了快速行动而不得不采取的手段。有一个说法，如果面条代码能够让我们行动更快，我们就会写出面条代码，不管是刀削面还是担担面。

我见过的优秀的程序员，无一例外，都对简洁代码有着偏执般的执着。甚至小到缩进空格该使用几个空格这样细枝末节的问题，都会严格地遵守编码的规范。乍一看，纠缠于缩进空格不是浪费时间吗？可是真相是，把小问题解决好，事实上节省了大量的时间。

这些对代码整洁充满热情的工程师，会对整个团队产生积极的、至关重要的影响。这种影响，不仅仅关乎到工程进展的速度，还关系到工程的质量。真正能够使得产品获得成功，甚至扭转科技公司命运的，不是关键时刻能够救火的队员，而是从一开始就消除了火灾隐患的队员。

## **简单直观减轻沟通成本**

简单直观的解决方案，有一个很大的优点，就是容易理解，易于传达。事情越简单，理解的门槛越低，理解的人越多，传达越准确。一个需要多人参与的事情，如果大家都能够清晰地理解这件事情，这就成功了一半。

我们不要忘了，客户也是一个参与者。简单直观的解决方案，降低了用户的参与门槛，减轻了学习压力，能够清晰地传递产品的核心价值，最有可能吸引广泛的用户。

## **简单直观降低软件风险**

软件最大的风险，来源于软件的复杂性。软件的可用性，可靠性，甚至软件的性能，归根到底，都是软件的复杂性带来的副产品。越复杂的软件，我们越难以理解，越难以实现，越难

以测量，越难以实施，越难以维护，越难以推广。如果我们能够使用简单直接的解决方案，很多棘手的软件问题都会大幅地得到缓解。

如果代码风格混乱，逻辑模糊，难以理解，我们很难想象，这样的代码会运行可靠。

## 该怎么做到简单直观？


如果我们达成了共识，要保持软件的简单直观，那么，我们该怎么做到这一点呢？最重要的就是做小事，做简单的事情。

### 使用小的代码块

做小事的一个最直观的体现，就是代码的块要小，每个代码块都要简单直接、逻辑清晰。整洁的代码读起来像好散文，赏心悦目，不费力气。

如果你玩过乐高积木，或者组装过宜家的家具，可能对“小部件组成大家具”的道理会深有体会。代码也是这样，一小块一小块的代码，组合起来，可以成就大目标。作为软件的设计师，我们要做的事情，就是识别、设计出这些小块。如果有现成的小块代码可以复用，我们就拿来用。如果没有现成的，我们就自己来实现这些代码块。


为了保持代码块的简单，给代码分块的一个重要原则就是，**一个代码块只做一件事情**。前面，我们曾经使用过下面的例子。这个例子中，检查用户名是否符合用户名命名的规范，以及检查用户名是否是注册用户，放在了一个方法里。

 复制代码


```
1 /**
2  * Check if the {@code userName} is a registered name.
3  *
4  * @return true if the {@code userName} is a registered name.
5  * @throws IllegalArgumentException if the {@code userName} is invalid
6  */
7 boolean isRegisteredUser(String userName) {
8     // snipped
9 }
```

如果单纯地从代码分块来看，还有优化的空间。我们可以把上述的两件事情，分别放到一个方法里。这样，我们就有了两个可以独立使用的小部件。每个小部件都目标更清晰，逻辑更

直接，实现更简单。

 复制代码

```
1 /**
2  * Check if the {@code userName} is a valid user name.
3  *
4  * @return true if the {@code userName} is a valid user name.
5  */
6 boolean isValidUserName(String userName) {
7     // snipped
8 }
```

 复制代码

```
1 /**
2  * Check if the {@code userName} is a registered name.
3  *
4  * @return true if the {@code userName} is a registered name.
5  */
6 boolean isRegisteredUser(String userName) {
7     // snipped
8 }
```

## 遵守约定的惯例

把代码块做小，背后隐含一个重要的假设：这些小代码块要容易组装。不能进一步组装的代码，如同废柴，没有一点儿价值。

而能够组装的代码，接口规范一定要清晰。越简单、越规范的代码块，越容易复用。这就是我们前面反复强调的编码规范。

## 花时间做设计

对乐高或者宜家来说，我们只是顾客，他们已经有现成的小部件供我们组合。对于软件工程师而言，我们是软件的设计者，是需要找出识别、设计和实现这些小部件的人。

识别出这些小部件，是一个很花时间的东西。

有的程序员，喜欢拿到一个问题，就开始写代码，通过代码的不断迭代、不断修复来整理思路，完成设计和实现。这种方法的问题是，他们通常非常珍惜自己的劳动成果，一旦有了成型的代码，就会像爱护孩子一般爱护它，不太愿意接受新的建议，更不愿意接受大幅度的修改。结果往往是，补丁擦补丁，代码难看又难懂。

有的程序员，喜欢花时间拆解问题，只有问题拆解清楚了，才开始写代码。这种方法的问题是，没有代码的帮助，我们很难把问题真正地拆解清楚。这样的方法，有时候会导致预料之外的、严重的架构缺陷。

大部分的优秀的程序员，是这两个风格某种程度的折中，早拆解、早验证，边拆解、边验证，就像剥洋葱一样。

拆解和验证，看起来很花时间。是的，这两件事情的确很耗费时间。但是，如果我们从整个软件的开发时间来看，这种方式也是最节省时间的。如果拆解和验证做得好，代码的逻辑就会很清晰，层次会很清楚，缺陷也少。

一个优秀的程序员，可能 80% 的时间是在设计、拆解和验证，只有 20% 的时间是在写代码。但是，拿出 20% 的时间写的代码，可能要比拿出 150% 时间写的代码，还要多，还要好。这个世界真的不是线性的。

有一句流传的话，说的是“跑得慢，到得早”。这句话不仅适用于健身，还适用于写程序。

## 借助有效的工具

我自己最常使用的工具，就是圆珠笔和空白纸。大部分问题，一页纸以内，都可以解决掉。当然，这中间的过程，可能需要一打甚至一包纸。

一旦问题有点大，圆珠笔和空白纸就不够用了。这时候，我们需要称手的工具，帮助我们记忆和思考。

现在我最喜欢的工具有思维导图、时序图和问题清单。在拆解问题时，思维导图可以帮助我厘清思路，防止遗漏。时序图可以帮助我理解关键的用例，勾画清楚各个部件之间的联系。而问题清单，可以记录下要解决和已经解决的问题，帮助我记录状态、追踪进度。

你最顺手的工具是什么？欢迎你分享在留言区，我们一起来学习。

## 小结

今天，我们主要聊的话题，就是做小事。我们工作生活中，一旦出现两种以上的竞争策略，要记住这个经过实践检验的理念：选择最简单，最直观的解决方案。

当然，我们遇到的不会总是简单的问题。如果把复杂的问题、大的问题，拆解成简单的问题、小的问题，我们就能够化繁为简，保持代码的整洁和思路的清晰。

## 一起来动手

通常一个用户登录的设计，需要输入用户名和密码。用户名和密码一起传输到服务器进行校验，授权用户登录。但现在有了更先进的设计。用户先输入用户名，用户名通过服务器检验，才能进一步输入密码，然后授权用户登录。

你愿不愿意分析一下，这种简单的流程变化，带来的收益？客户端和服务端接口代码，大致应该是什么样子的？你使用了什么工具来分析这些问题？

欢迎你在留言区讨论上面的问题，我们一起来看看这种简单的变化可以带来什么样的好处。



# 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。



上一篇 19 | 怎么避免过度设计？

下一篇 21 | 怎么设计一个简单又直观的接口？

## 精选留言 (12)

写留言



唐名之

2019-02-18

5

1: 首先会校验用户名，可能是为了检测当前用户登录环境是否安全；  
2: 通过先检测登录环境是否安全会减少密码泄露风险；  
3: 从接口设计，第一步用户检测服务端接口应该会返回检测成功与否标识和唯一的会话标识，客户端待用户输入密码，会将 用户名 密码 会话唯一标识 提交到服务端，服务端会将三者一起作为校验；

展开

作者回复: 安全是这个流程改进最大的好处之一。



hua168

2019-02-20

2

像京东架构那样，他们最小单位是模块，基础模块的话，很多模块都会调用，以垂直划分的话，这类模块放在最下面，比如数据库模块，有一说法“基础模块下沉” .....  
几个最基础的模块又组成一个大一点的基础模块，放在中间位置...  
最顶的是复用最少的，但由中间层的几个基础模块组成.....  
一句话:复用越多的模块越放下面，复用越多越是基础模块.....那不是“基础模块下沉，独...  
展开

作者回复: 看起来像是说依赖关系，上层依赖下一层，复用模块不依赖上一层的，可以考虑放到下层的模块，这样复用更方便更多。



Junix

2019-02-18

2

文中的问题：

1、先检验用户名是否存在，其一会让输错的用户省去输入密码的时间，尤其是密码复杂的。其二可以帮用户定位出错位置，用户名错误或密码错误。输入用户名后，输入密码前

这段时间里用户能直接了当的看到用户名是否错误。输入密码后再报错，就能知道是密码错误，直接帮用户进行了校验，定位了出错的位置。带来的收益就是直观和简洁，与第...  
展开 ▾

作者回复: 定位错误是其中的收益之一。如果1/10的用户输入用户名错误，你可以简单估算、揣摩一下实现得当的服务器的性能提升比例以及用户的体验提升程度。

现在的界面，也有的不显示密码输入框，直到用户名校验通过，才显示密码输入框。也有的使用两个页面，第一个页面用户名，第二个页面密码。



北风一叶

2019-04-02

👍 1

代码块越小，越容易复用，这个和“重构改善即有代码设计”里的观点一致

展开 ▾



lamNigel

2019-03-05

👍 1

文中的问题想到一点，可以防止撞库式的外部冲击对最底层的用户名密码验证模块产生直接冲击，如果100次尝试，第一道就拦掉了90次就产生了直接受益，这种时候还可以在第一时间拦截用户名校验处对这种ip进行黑名单处理

展开 ▾

作者回复: 我一直想在留言区看到有人讨论这个方案的安全改进。谢谢你分享了这个想法!



aguan(^·...

2019-02-19

👍 1

重新阅读了一遍文章，发现设计成两个接口的还有一个好处是接口细分了更容易复用，比如修改密码的时候可以复用登录时密码检验的接口

作者回复: 嗯，这也是一个优点。



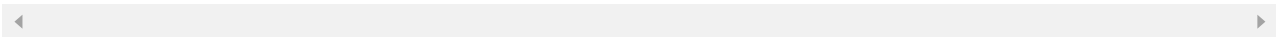


苦行僧  
2019-02-19



只有持续不断的重构review才能将代码改进到最好，很多写代码的完成业务功能就完事，具有精益思维是简单的前提

作者回复: 你说的精益思维是持续改进的意思吗？我比较倾向于一开始就选择简单的方案，最核心的需求，然后再把这些简单的东西做好，持续改进。



Sisyphus2...

2019-05-23



分离用户名和密码的验证：

1. 从产品层面模块化，如果有问题能清晰指出哪里的问题
2. 用户体验升级，如果用户名不存在不需要输入密码
3. 服务端更灵活，如果用户量巨大，可以做一个用户名的缓存，而不用每次登陆都必须连接数据库验证用户密码正误...

展开 ∨



木白

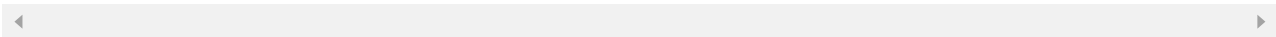
2019-04-30



有的代码是一个独立的功能，但是只在一个地方使用，没有其他地方需要复用它，这种时候就会失去把它拆成一个单独方法的动力。想请教下这种只用一次的“功能”还需要写成独立的代码块吗？

展开 ∨

作者回复: 看代码逻辑和长度吧。如果长度段，写在使用它的方法内就好了；如果长度长，逻辑独立，拆解成一个内部私有的方法，更容易理解和阅读。



hua168

2019-02-19



一个项目有n多的代码，一个代码块只做一件事情的话，那么很多重复用的代码，是不是可以变成基础模块？

我看到有人说“基础模块下沉，方便复用”，那在中间层和上层的模块呢？有什么原则之类吗？

作者回复: 每一层的代码, 都要考虑复用。也许, “下沉” 说的是复用代码剥离出来, 当作通用的类库? 我不太了解这个说法。



空知

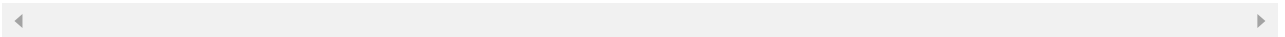
2019-02-18



想问下怎么通过用户名检测登录环境是否安全的?

展开 ∨

作者回复: 确实有空间来检测登陆环境。比如说, 通过检查该用户名是否正在使用常用的设备, 来决定是否启用更多的校验, 比如验证码、检查是否人工操作等。



aguan(^·...

2019-02-18



除了用户体验和安全性的好处, 想不到还有哪些性能上的提升了。分不分两个接口, 总的处理逻辑都是不变的, 挺想知道对性能上有哪些提升, 求大神们指点一二。

展开 ∨

作者回复: 分不分两个接口, 总体逻辑可以不变, 也可以改变。性能提升不太容易想的到, 理一理客户端和服务端的代码的什么样, 想一想可以怎么变化, 意外情况怎么处理。

