



下载APP



01 | 缺乏业务含义的命名：如何精准命名？

2020-12-28 郑晔

代码之丑

[进入课程 >](#)



讲述：郑晔

时长 12:12 大小 11.18M



你好，我是郑晔。


讲写代码的书通常都会从命名开始讲，《[🔗 程序设计实践](#)》如此，《[🔗 代码整洁之道](#)》亦然。所以，我们这个讲代码坏味道的专栏，也遵循传统，从命名开始讲。

不过，也许你会说：“我知道，命名不就是不能用 `abcxyz` 命名，名字要有意义嘛，这有什么好讲的。”然而，即便懂得了名字要有意义这个道理，很多程序员依然无法从命名的泥潭中挣脱出来。



不精准的命名

我们先来看一段代码：

 复制代码

```
1 public void processChapter(long chapterId) {  
2     Chapter chapter = this.repository.findByChapterId(chapterId);  
3     if (chapter == null) {  
4         throw new IllegalArgumentException("Unknown chapter [" + chapterId + "]");  
5     }  
6  
7     chapter.setTranslationState(TranslationState.TRANSLATING);  
8     this.repository.save(chapter);  
9 }
```

这是一段看上去还挺正常的代码，甚至以很多团队的标准来看，这段代码写得还不错。但如果我问你，这段代码是做什么的。你就需要调动全部注意力，去认真阅读这段代码，找出其中的逻辑。经过阅读我们发现，这段代码做的就是将一个章节的翻译状态改成翻译中。

问题来了，为什么你需要阅读这段代码的细节，才能知道这段代码是做什么的？

问题就出在函数名上。这个函数的名字叫 `processChapter`（处理章节），这个函数确实是在处理章节，但是，这个名字太过宽泛。如果说“将章节的翻译状态改成翻译中”叫做处理章节，那么“将章节的翻译状态改成翻译完”是不是也叫处理章节呢？“修改章节内容”是不是也叫处理章节呢？换句话说，如果各种场景都能够叫处理章节，那么处理章节就是一个过于宽泛的名字，没有错，但不精准。

这就是一类典型的命名问题，从表面上看，这个名字是有含义的，但实际上，它并不能有效地反映这段代码的含义。如果说我在做的是一个信息处理系统，你根本无法判断，我做的是一个电商平台，还是一个图书管理系统，从沟通的角度看，这就不是一个有效的沟通。要想理解它，你需要消耗大量认知成本，无论是时间，还是精力。

命名过于宽泛，不能精准描述，这是很多代码在命名上存在的严重问题，也是代码难以理解的根源所在。

或许这么说你的印象还是不深刻，我们看看下面这些词是不是经常出现在你的代码里：`data`、`info`、`flag`、`process`、`handle`、`build`、`maintain`、`manage`、`modify` 等等。这些名字都属于典型的过于宽泛的名字，当这些名字出现在你的代码里，多半是写代码的人当

时没有想好用什么名字，就开始写代码了。我相信，只要稍微仔细想想，类似的名字你一定还能想出不少来。

回到前面那段代码上，如果它不叫“处理章节”，那应该叫什么呢？首先，**命名要能够描述出这段代码在做的事情**。这段代码在做的事情就是“将章节修改为翻译中”。那是不是它就应该叫 `changeChapterToTranslating` 呢？

不可否认，相比于“处理章节”，`changeChapterToTranslating` 这个名字已经进了一步，然而，它也不算是一个好名字，因为它更多的是在描述这段代码在做细节。我们之所以要将一段代码封装起来，一个重要的原因就是，我们不想知道那么多的细节。如果把细节平铺开，那本质上和直接阅读代码细节差别并不大。

所以，**一个好的名字应该描述意图，而非细节**。

就这段代码而言，我们为什么要把翻译状态修改成翻译中，这一定是有原因的，也就是意图。具体到这里的业务，我们把翻译状态修改成翻译中，是因为我们在这里开启了一个翻译的过程。所以，这段函数应该命名 `startTranslation`。

[复制代码](#)

```
1 public void startTranslation(long chapterId) {
2     Chapter chapter = this.repository.findByChapterId(chapterId);
3     if (chapter == null) {
4         throw new IllegalArgumentException("Unknown chapter [" + chapterId + "]");
5     }
6
7     chapter.setTranslationState(TranslationState.TRANSLATING);
8     this.repository.save(chapter);
9 }
```

用技术术语命名

我们再来看一段代码：

[复制代码](#)

```
1 List<Book> bookList = service.getBooks();
```

可以说这是一段常见得不能再常见的代码了，但这段代码却隐藏另外一个典型得不能再典型的问题：**用技术术语命名**。


这个 bookList 变量之所以叫 bookList，原因就是它声明的类型是 List。这种命名在代码中几乎是随处可见的，比如 xxxMap、xxxSet。

这是一种不费脑子的命名方式，但是，这种命名却会带来很多问题，因为它是一种基于实现细节的命名方式。

我们都知道，编程有一个重要的原则是面向接口编程，这个原则从另外一个角度理解，就是不要面向实现编程，**因为接口是稳定的，而实现是易变的**。虽然在大多数人的理解里，这个原则是针对类型的，但在命名上，我们也应该遵循同样的原则。为什么？我举个例子你就知道了。

比如，如果我发现，我现在需要的是一个不重复的作品集合，也就是说，我需要把这个变量的类型从 List 改成 Set。变量类型你一定会改，但变量名你会改吗？这还真不一定，一旦出现遗忘，就会出现一个奇特的现象，一个叫 bookList 的变量，它的类型是一个 Set。这样，一个新的混淆就此产生了。

那有什么更好的名字吗？我们需要一个更面向意图的名字。其实，我们在这段代码里真正要表达的是拿到了一堆书，所以，这个名字可以命名成 books。


 复制代码

```
1 List<Book> books = service.getBooks();
```

也许你发现了，这个名字其实更简单，但从表意的程度上来说，它却是一个更有效

虽然这里我们只是以变量为例说明了以技术术语命名存在的问题，事实上，**在实际的代码中，技术名词的出现，往往就代表着它缺少了一个应有的模型**。

比如，在业务代码里如果直接出现了 Redis：

 复制代码

```
1 public Book getByIsbn(String isbn) {  
2     Book cachedBook = redisBookStore.get(isbn);  
3     if (cachedBook != null) {  
4         return cachedBook;  
5     }  
6  
7     Book book = doGetByIsbn(isbn);  
8     redisBookStore.put(isbn, book);  
9     return book;  
10 }
```

通常来说，这里真正需要的是一个缓存。Redis 是缓存这个模型的一个实现：

[复制代码](#)

```
1 public Book getByIsbn(String isbn) {  
2     Book cachedBook = cache.get(isbn);  
3     if (cachedBook != null) {  
4         return cachedBook;  
5     }  
6  
7     Book book = doGetByIsbn(isbn);  
8     cache.put(isbn, book);  
9     return book;  
10 }
```

再进一步，缓存这个概念其实也是一个技术术语，从某种意义上说，它也不应该出现在业务代码中。这方面做得比较好的是 Spring。使用 Spring 框架时，如果需要缓存，我们通常是加上一个 Annotation（注解）：

[复制代码](#)

```
1 @Cacheable("books")  
2 public Book getByIsbn(String isbn) {  
3     ...  
4 }
```

程序员之所以喜欢用技术名词去命名，一方面是因为，这是大家习惯的语言，另一方面也是因为程序员学习写代码，很大程度上是参考别人的代码，而行业里面优秀的代码常常是一些开源项目，而这些开源项目往往是技术类的项目。**在一个技术类的项目中，这些技术术语其实就是它的业务语言。但对于业务项目，这个说法就必须重新审视了。**

如果这个部分的代码确实就是处理一些技术，使用技术术语无可厚非，但如果是在处理业务，就要尽可能把技术术语隔离开来。


用业务语言写代码

无论是不精准的命名也好，技术名词也罢，归根结底，体现的是同一个问题：对业务理解不到位。

我在《[🔗 10x 程序员工作法](#)》专栏中曾经说过，**编写可维护的代码要使用业务语言**。怎么才知道自己的命名是否用的是业务语言呢？一种简单的做法就是，把这个词讲给产品经理，看他知不知道是怎么回事。

从团队的角度看，让每个人根据自己的理解来命名，确实就有可能出现千奇百怪的名字，所以，一个好的团队实践是，**建立团队的词汇表**，让团队成员有信息可以参考。


团队对于业务有了共同理解，我们也许就可以发现一些更高级的坏味道，比如说下面这个函数声明：

 复制代码

```
1 public void approveChapter(long chapterId, long userId) {  
2     ...  
3 }
```

这个函数的意图是，确认章节内容审核通过。这里有一个问题，chapterId 是审核章节的 ID，这个没问题，但 userId 是什么呢？了解了一下背景，我们才知道，之所以这里要有一个 userId，是因为这里需要记录一下审核人的信息，这个 userId 就是审核人的 userId。

你看，通过业务的分析，我们会发现，这个 userId 并不是一个好的命名，因为它还需要更多的解释，更好的命名是 reviewerUserId，之所以起这个名字，因为这个用户在这个场景下扮演的角色是审核人（Reviewer）。

 复制代码

```
1 public void approveChapter(long chapterId, long reviewerUserId) {  
2     ...  
3 }
```

从某种意义上来说，这个坏味道也是一种不精准的命名，但它不是那种一眼可见的坏味道，**而是需要在业务层面上再进行讨论**，所以，它是一种更高级的坏味道。

我初入职场的時候，有一次为一个名字陷入了沉思，一个工作经验丰富的同事对此的评价是：你开始进阶了。确实，能够意识到自己的命名有问题，是程序员进阶的第一步。

总结时刻

我们今天讲了两个典型的命名坏味道：

不精准的命名；

用技术术语命名。

命名是软件开发中两件难事之一（另一个难事是缓存失效），不好的命名本质上是增加我们的认知成本，同样也增加了后来人（包括我们自己）维护代码的成本。

好的命名要体现出这段代码在做的事情，而无需展开代码了解其中的细节，这是最低的要求。再进一步，好的命名要准确地体现意图，而不是实现细节。更高的要求是，用业务语言写代码。

至此，我们已经对命名有了一个更深入的认识。下一讲，我们来说说国外那些经典的讲编码的书都不曾覆盖到的一个话题：英文命名。

如果今天的内容你只能记住一件事，那请记住：**好的命名，是体现业务含义的命名。**

划重点 01

坏味道：缺乏业务含义的命名

错误命名

1. 宽泛的命名。
2. 用技术术语命名。

命名遵循的原则

1. 描述意图，而非细节。
2. 面向接口编程，接口是稳定的，实现是易变的。
3. 命名中出现技术名词，往往是它缺少了一个模型。
4. 使用业务语言。

记住一句话

好的命名，是体现业务含义的命名。



思考题

前面我们提到了一些代码中常见的不精准的命名所用的词汇，你还能想到哪些词呢？欢迎在留言区分享你的想法。也欢迎你把这节课分享给你身边对命名问题感到困惑的朋友。

感谢阅读，我们下一讲再见！

参考资料: [🔗 你的代码为谁而写？](#)

提建议

© 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。

上一篇 课前热身 | 这些需求给到你，你会怎么写代码？

下一篇 02 | 乱用英语：站在中国人的视角来看英文命名

精选留言 (23)

写留言



刘大明

2020-12-28

老师提的命名不规范，在项目中全中。
还有项目总监写的这种代码
`aaaaaa.updateFlight(airOrder, airOrderOld);`
各种不规范命名。难受

展开 ∨

作者回复: aaaaaa还是突破底线了。

2

8



Jxin

2020-12-29

- 1.实际工作时，去抓别人的命名规范其实挺不舒服的，显得自己爱计较，而且在缺少共同认知，一般人家也觉得你爱计较。
- 2.命名其实就是对抽象的定义。不能描述业务含义的命名往往是由于抽象的角度不正确或不明确引起的。设计建模时我们会说，业务驱动,认知先行。这里的认知先行就是要先建立与业务侧达成共识的模型（可以认为就是文中让产品看看能不能理解）。然后基于这个...
展开 ∨

作者回复: 坚持做正确的事情有难度，不坚持却难以提升。

1

5



捞鱼的搬砖奇

2020-12-29

老师好，我又又来了。

18年末，我看了你的10X 工作法专栏，里面有提到过“代码的坏味道”碰巧那时候我刚读完重构第一版，对你专栏的观点非常认同，几个月之后重构出了第二版，一半是好奇，一半是自己对优秀代码的向往把书买回来读完了。几个月前我看完了软件设计之美。这样一想三年过去了，自己在代码量有了更多的积累，每次回过去看那些重构的原则，都会...
展开 ∨

作者回复: 欢迎回来！

进步都一点一点来的，没有谁一开始就很牛。

3

3



huaweichen

2020-12-30

为了命名，和同事展开了很多争论：

原命名为\$personsArray。

我的任务是重构。

我在重构过程中，我把变量换为Collection类型，变量名改为\$persons。

...

展开 ∨

作者回复: 当它把类型从 Array 改成 Collection，就需要改一次变量名，那以后同样的事情还是会发生的。

同样，这里的复数实际上也可以里面只有一个，这并不是问题。

你可以看一下我在文章结尾附上的资料，这是我在《10x 程序员工作法》关于命名的介绍。



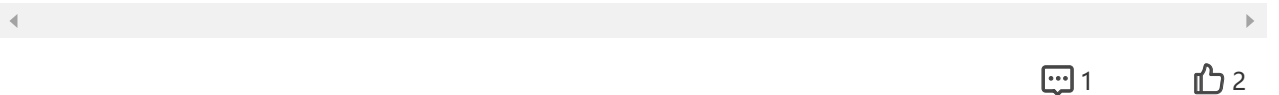
Demon.Lee

2020-12-30

老师，xxxMap这种命名表示映射关系，比如：书id与书的映射关系，不能命名为bookId Map么？一时没想到好的命名，老师能给点建议么

展开 ∨

作者回复: 按照我习惯的约定，Map 表示的是一个数据结构，而映射关系我会写成 Mapping。



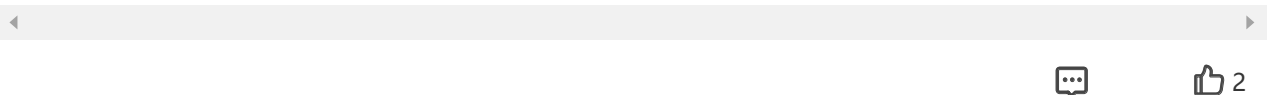
木子轩

2020-12-29

看完后我去review了下我的代码，确实有的地方命名太过宽泛。之前也遇到过返回值类型改了之后，下面的变量名很尴尬的问题。以后写代码会注意的到了。

展开 ∨

作者回复: 有所得，没白学。



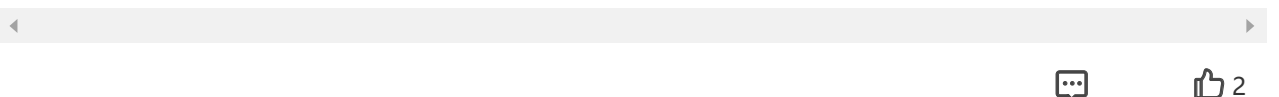
webmin

2020-12-29

听完老师的课，对有意义的名字的理解有进了一步，之前只做到不用abcd等来命名，今天学到了名字要根据场景取一个有业务含义的名字。表明意图的名字可以让Review代码的人先了解意图，再结合情况来看是否要了解细节。

展开 ∨

作者回复: 有直接的收获就好，专栏没白学。



瓜瓜

2020-12-28

果断拿下

展开 ∨



2



adang

2021-01-02

进入程序员这个行业，第一个使用的框架就是Rails，Rails在很多地方做了很好的约定，例如：Controller名字要用复数，Model名字要用单数，相对应的数据库表名名字要用复数等。老师课程中举的变量名bookList的例子，在Rails中约定的命名就是books。

现在已经不再使用ROR了，但是在动手之前还是会认真思考，尤其是有多种方式可以选择，拿不准要采用哪种的时候会想，如果这种场景在Rails下会怎样处理，这个时候通常...

展开 ∨

作者回复: 从最佳实践的角度来说，Rails一直是表现优异的，因为它的设计者是行业中的佼佼者。所以，我在《软件设计之美》中，专门把它拿出来讲。



1



Sinvi

2021-01-01

考虑业务相关的话有时候命名会比较长。。。

展开 ∨

作者回复: 如果考虑到在特定的上下文里，有些名字就可以短一些了。



1



冯将

2020-12-30

```
@GetMapping("getTotalSettlementInfoByYear")
```

```
@ApiOperation("公司结算信息按年求和")
```

```
public Result<List<RepMonthCompanyDTO>> getTotalSettlementInfoByYear(@RequestParam String year) {
```

```
    List<RepMonthCompanyDTO> list = repMonthCompanyService.getTotalSettle...
```

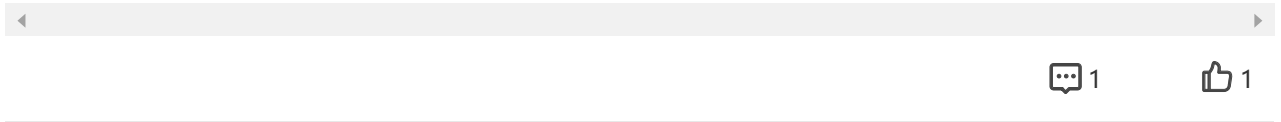
展开 ∨

作者回复: 名字长不是问题，问题是表达是否清晰，像repMonthCompanyService这个名字，是不太容易一眼看出来含义的。

另外，传给 service 的参数是一个字符串，这个从逻辑上是有问题的，没有进行参数的校验。后面的内容也会讲到，这个做法是一种缺乏封装的表现。

变量名是 list，按照这一讲的说法是用技术术语在命名。

再有，这个 URI 是 getTotalSettlementInfoByYear，这是不符合 REST 的命名规范的，比如，动词不应该出现在 URI 里，分词应该是“-”，byYear 实际上是一个过滤条件等等。



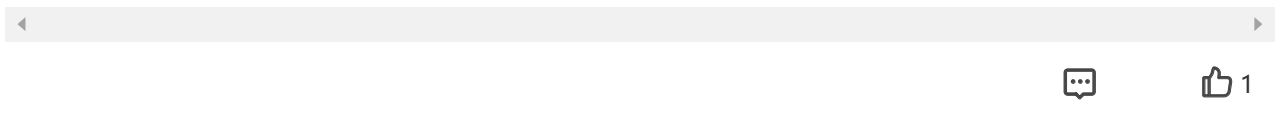
topsion

2020-12-30

感谢大大用我的代码做经典案例。😊😊😊😊

展开 ∨

作者回复: 加油!



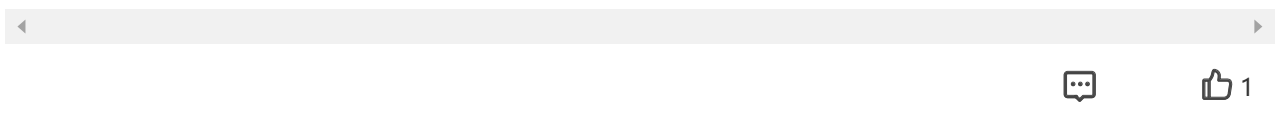
Geek_3b1096

2020-12-30

写了很多processXXXX

展开 ∨

作者回复: 多么痛的领悟



Alexdown

2020-12-29

唉，啥也白说了，都是眼泪啊.....

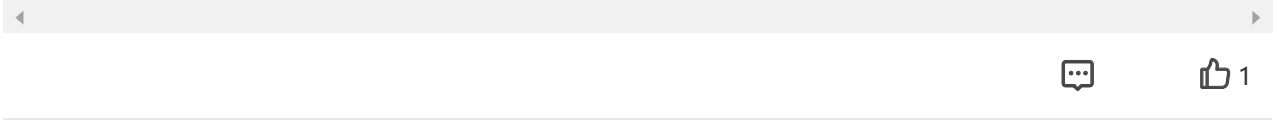
刚写代码时，老鸟通常会对菜鸟说教：你不要期望一次将代码写好，得注重健壮性（鲁棒性）、可读性、可维护性、可扩展性，多思考、常重构....多看书去看看《重构》《代码整洁之道》等等。然而你看他的提交记录除了改bug就没重构过！再看看他写的代码（后...

展开 ∨

作者回复: 先对你的处境表示一下同情。

分层命名一般是团队的习惯，像我现在的团队，一般是用 resource/application/service/repository 来命名，实际上，这里的 application 对应着 DDD 中的应用服务，service 对应着领域服务，在很多团队里，这些都放到了 service 里面。

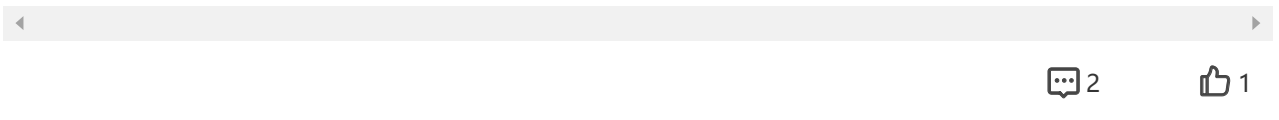
也有的团队 resource 叫 controller，repository 叫 mapper 或 dao。具体叫哪个，按照团队的约定就好。

**明**

2020-12-28

老师说的初入职场思考命名那块，深有同感，我现在就发现不会写代码了 不知道怎么样才能写好 也不知道这些对不对 导致写的特别慢 不知道啥时候能度过这个迷茫期

作者回复: 坚持以这个标准，多看一些代码，积累一些经验。

**小袁**

2020-12-28

我曾见在一个很长的代码中看到。catch(exception ee1)~atch(exception ee22)

作者回复: 22，这个还是挺有勇气的。

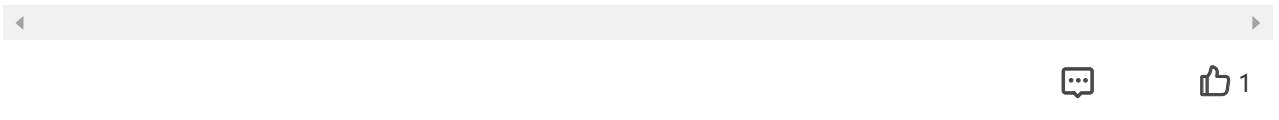
**00舟州**

2020-12-28

我们的代码里恰好就有类似userId的问题，实际业务是创建者和修改者，可惜方法都给取名uld了，虽然有方法名，但听完课感觉还是应该统一下。

展开 ∨

作者回复: 能直接用起来，真好。

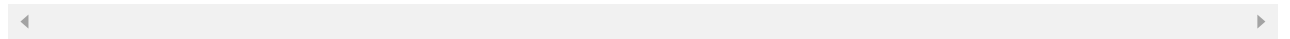
**封志强**

2020-12-28

老师，来了

展开 ▾

作者回复: 欢迎加入！



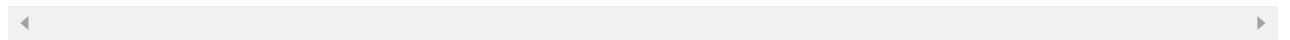
gker

2021-01-23

如果是map的命名，老师的意思是以mapping结尾吗？还有如果是那种复杂结构的map，类似于，map<string，map<>>，这种怎么命名好一点呢？请指教

展开 ▾

作者回复: 通常来说，在写代码的过程中，map 是一种数据结构，而 mapping 是映射的意思，英文上是不一样的。重点不是 map 对象该如何命名，而是业务上是什么含义。



Hobo

2021-01-11

如果叫reviewerId是不是更好一点

展开 ▾

作者回复: 你说得有道理，是可以的。

