##**11.6** 创建具备可读性的短名字

从某种程度上说，要求使用短变量名是早期计算的遗留物。早期语言，如汇编、一般的 `Basic` 和 `Fortran` 都把变量名的长度限制在 2 到 8 个字符，并要求程序员创建简短的名字。早期的计算科学更多的痛数学联系在一起，并要求程序员创建简短的名字。早期的计算科学更多的同数学联系在一起并大量使用求和及其他等式中的 i、j 和 k 等符号。而在现代语言如 `C++` 、 `Java` 和 `Visual Basic` 里面，实际上你可以创建任何长度的名字；几乎没有任何理由去缩短具有丰富含义的名字。如果环境真的要求你创建简短的名字，请注意有些缩短名字的方法要好于其他的方法。你可以通过消除冗余的单词、使用简短的同义词以及诸多缩写策略中的任何一种来创建更好的短变量名。熟悉多种缩写技巧会很有用，因为没有哪种方法能够适用于所有的情况。

###缩写的一般指导原则

下面是几项用于创建缩写的指导原则。其中的一些原则彼此冲突，所以不要试图同时应用所有原则。

- 使用标准的缩写（列在字典中的那些常见缩写）。
- 去掉所有非前置元音。（computer 变 cmptr，screen 变成 scrn，apple 变成 appl，integer 变成 intgr）
- 去掉虚词 and，or，the等。
- 使用每个单词的第一个或前几个字母。
- 统一地在每个单词的第一、第二或者第三个（选择最合适的一个）字母后截断。
- 保留每个单词的第一和最后一个字母。
- 使用名字中的每一个重要单词，最多不超过三个。
- 去除无用的后缀 —— ing，ed 等。
- 确保不要改变变量的含义。
- 反复使用上述技术，直到你把每个变量名的长度缩减到了 8 到 20 个字符，或者达到你所用的编程语言对变量名的限制字符数。

###语音缩写 有些人倡导基于单词的发音而不是拼写来创建缩写。于是 skating 就变成了 sk8ing，highlight 变成了 hilite，before 变成了 b4，execute 变成了 xqt，诸如此类。这样做很像是要人去猜出个性化汽车牌照的意思，我不倡导这么做。作为一项练习，请猜猜下面这些名字个表示什么：

ILV2SK8 XMEQWK S2DTM8O NXTC TRMN8R

###有关缩写的评论

在创建缩写的时候，会有很多的陷阱在等着你。下面是一些能够避免犯错的规则。

- 不要用从每个单词中删除一个字符的方式来缩写。 键入一个字符算不上是什么额外工作，而节省一个字符带来的便利却很难抵消由此而造成的可读性损失。这就像日历中的 “Jun” 和 “Jul”。你只有在非常着急的情况才有必要把 June 拼成 “Jun”。对于大多数删除一个字母的做法而言，你很难回忆起自己是不是删了一个字符。所以要么删除不止一个字符，要么就把单词拼写完整。
- 缩写要一致 应该一直使用相同的缩写。例如，要么全用 Num，要么全用 No，不要两个都用。与之类似，不要在一些名

字里缩写某个单词而在其他名字里不缩写。比如，不要在有些地方使用完整的单词 Number，同时在其他地方使用 Num 缩写。

- 创建你能读出来的名字 用 xPstn,用 needsCompu 而不用 ndsCmptg。此处可以借助电话来测试 —— 如果你无法在电话中向他人读出你的代码，就请重新给变量起一个更清晰的名字吧。
- 避免使用容易看错或者读错的字符组合 为了表示 B 的结尾，ENDB 要比 BEND 更好。如果你使用了一种好的分隔技术，那么就不需要这一条原则，因为 B_END、BEnd 或者 b_end 都不会被读错。
- 使用辞典来解决命名冲突 创建简短名字会带来的一项麻烦技术命名冲突 —— 缩写后名字相同。例如，如果命名长度被限制为 3 个字符，并且你需要在程序中的同一代吗使用 fired 和 full revenue disbursal，你可能会不经意地把缩写都写出了 frd。 避免命名冲突的一种简单做法是使用含义相同的不同单词，这样一来，有一部辞典就显得很方便。在本例中，可以用 dismissed 来代替 fired，以及用 complete revenue disbursal 来代替 full revenue disbursal。这样，3 个字母的缩写就分别变成了 dsm 和 crd，从而消除了命名冲突。
- 在代码里用缩写对照表解释极短的名字的含义 当编程语言只允许用非常短的名字的时候，增加一张缩写对照表来为用户提升更多的变量含义。把该表格作为注释加到一段代码的开始。下面是一个例子：

Fortran 示例：良好的名字对照表

```
Tranalation Table
Variable Meaning
......   ......
XPOS x-Coordinate Position(in meters)
YPOS y-Coordinate Position(in meters)
NDSCMP Needs Computing (= 0 if no computation is needed;
                        = 1 if  computation is needed)
PTGTTL Point Grand Total
PTVLMX Point Value Maxinum
PSCRMX Possible Score Maxinum
```

你可能会认为这种方法以及过时了，但是在 2003 年中期，我与一家客户合作，该客户有上万行用 `RPG` 语言写成的、变量名被限制在6个字符以内的代码。这些要求极短变量名的问题仍然时不时地出现。

在一份项目级的“标准缩写”文档中说明所有的缩写 代码中的缩写会带来的两种常见风险。

- 代码的读者可能不理解这些缩写。
- 其他程序员可能会用多个缩写来代表相同的词，从而产生不必要的混乱。

为了同时解决这两个潜在的问题，你可以创建一份“标准缩写”文档来记录项目中用到的全部编码的缩写。这份文档既可以是文字处理程序的文档，也可以是电子表格文档。在很大的项目里，它还可以是一个数据库。这份文档应签入（check in）到版本控制系统里，当任何人于任意时间在代码里创建一种新的缩写把它签出（check out）来修改。文档中的词条应该按照完整单词排序，而不是按照缩写排序。

这看上去可能显得非常麻烦，但是除了开始的一点额外工作，它事实上是建立了一种在项目中有效地使用缩写的机制。通过对所有用到的缩写加以说明，就解决了上面描述的两种常见风险中的第一种。程序员如果不费力把标准缩写文档从版本控制系统中 check out、输入新的缩写并把它 check in 回去，就不能常见一个新的缩写。这是件好事。它表明，只有当一个缩写在代码中应用非常广泛，程序员不惜花上很多精力来为它编写缩写文档，这一缩写才的的确确应当被创建。

这种方法通过降低程序员创建多余的缩写的缩写的可能性，从而解决了第二种风险。想创建缩写的程序员会把缩写文档 check out 并输入新的缩写。如果他想缩写的单词已经有了缩写，该程序员就会注意到它，并且去使用该现有的缩写而不创建一个新的。

本原则中体现出来的核心问题，是方便编写代码同方便阅读两种理念之间的差异。上面的方法很明显会带来代码编写时的麻烦，但是程序员们在整个项目生命周期里会把更多的时间花在阅读代码而不是编写代码之上。这种方法提高了阅读代码的方便性。当一个项目尘埃落定之后，它可能还会提高编写代码的方便性。

记住，名字对于代码读者的意义要比对作者更重要 去读一读你自己写的并且至少六个月诶看过的代码，注意哪些名字是你需要花功夫才能理解其含义的。应下决心改变导致这种混乱的做法。