Chapter 11 变量名的力量（The Power Of Variable Names）

11.1 选择好变量名的注意事项（Considerations in Choosing Good Names）

类似 `x`， `xx`， `x1` 之类的变量名是糟糕的变量名，而类似 `balance`， `lastPayment`， `newPurchases` 之类的变量名是好的变量名。好的变量名是可读的，易记的和恰如其分的。

- 最重要的命名注意事项（**The Most Important Naming Considerations**）

  变量名要完全、准确地描述出该变量所代表的事物，对变量的描述就是最佳的变量名。好的变量名示例： `checkTotal`， `velocity`， `trainVelocity`， `currentDate`， `linesPerPage`。

- 以问题为导向（**Problem Orientation**）

  好记的名字反映的通常是问题，而不是解决方案。如果一个名字反映了计算的某些方面而不是问题本身，那么它反映的就是 `how` 而非 `what` 了。应该在名字中反映出问题本身。例如在财务软件中， `sum` 比 `calcVal` 更好。

- 最适当的名字长度（**Optimum Name Length**）

  变量平均名字在 `8` 到 `20` 个字符的时候调试比较容易。长度较合适的名字示例： `numTeamMembers`， `teamMemberCount`， `seatCount`， `pointsRecord`。

- 作用域对变量名的影响（**The Effect of Scope on Variable Names**）

  短的变量名（例如： `i`）代表一个临时的数据（循环计数器或者下标），其作用域非常有限。细心的程序员避免使用短的变量名。应对位于全局命名空间中的名字加以限定词，在 C++ 和 C# 里，可以使用 `namespace` 关键字来划分全局命名空间，在 Java 中，可以通过使用 package 达到同样的目的。对不支持命名空间或者包的语言，可以使用命名规则来划分全局命名空间，例如增加前缀（ `uiEmployee`， `dbEmployee` 等）。

- 变量名中的计算值限定词（**Computed-Value Qualifiers in Variable Names**）

  对于表示计算结果的变量，将总额，平均值，最大值（ `Total`， `Sum`， `Average`， `Max`， `Min`， `Record`， `String`， `Pointer` ）这样的限定词加到名字的最后。这样可以突出为变量赋予主要含义的部份（位于变量前部），也可以使变量名的规则统一，提高可读性。一个例外是：将 `num` 放置在变量的开始位置代表总数，而放置在变量结束位置代表下标。这样的情况应使用 `Count` （或者 `Total` ）以及 `Index` 来避开这个问题（例如： `customerCount` 及 `customerIndex` ）。

- 变量名中的常用对仗词（**Common Opposites in Variable Names**）

  在命名规则中应用准确的对仗词，将有助于变量的理解和记忆。常用的对仗词：

  - `begin / end`
  - `first / last`
  - `min / max`
  - `next / previous`
  - `old / new`
  - `source / target`
  - `up / down`

11.2 为特定类型的数据命名（Naming Specific Types of Data）

为特定类型的数据命名时，需要作出特殊的考虑。本节讲述与循环变量、状态变量、临时变量、布尔变量、枚举类型和具名常量有关的考虑事项。

- 为循环下标命名（**Naming Loop Indexes**）
  - 约定俗成的简单循环变量名：`i`，`j`，`k`
  - 若变量要在循环之外使用，应使用描述性较好的循环变量名（例如：`recordCount`）
  - 若使用嵌套循环，应给循环变量赋予更长的名字以提高可读性（例如：`score[teamIndex][eventIndex]` 比 `score[i][j]` 更清晰）。这有助于避免下标串话（index cross-talk）问题。

- 为状态变量命名（**Naming Status Variables**）

  状态变量用于描述程序的状态。应该为状态变量取一个比 `flag` 更好的名字，因为我们看不出 `flag` 是做什么的。标记应该用枚举类型，具名常量，或者用作具名常量的全局变量来对其赋值及比较。例如：`characterType = CONTROL_CHARACTER` 比 `statusFlag = 0x80` 更有意义。

- 为临时变量命名（**Naming Temporary Variables**）

  临时变量用于存储计算的中间结果，作为临时占位符，以及存储内务管理（housekeeping）值。但其实从某种角度来看，程序中的大多数变量都可以算是临时性的。我们不应该使用不提供任何信息的临时变量名（例如：`x`，`temp`），而应该使用具有描述性的真正的变量来替代“临时”变量。

- 为布尔变量命名（**Naming Boolean Variables**）

  应当给布尔变量赋予隐含**“真/假”**含义的名字。这些名字（例如：`done`）的状态要么是 `true`，要么是 `false`。而 `status` 等类型的名字没有明确的 `true` 或者 `false` 的概念。

  有些人喜欢在布尔变量名前加上 `is`，优点是避免了使用模糊不清的名字（例如：`isStatus`），缺点是降低了简单逻辑表达式的可读性（`is (isFound)` 的可读性要差于 `if (found)`）。

  应当使用肯定的布尔变量名，例如 `found`，否定的名字难以阅读（例如：`if not notFound`）。

  一些典型的布尔变量名：

  - `done`
  - `error`
  - `found`
  - `success / ok`

- 为枚举类型命名（**Naming Enumerated Types**）

  可以为枚举类型使用前缀命名约定（例如：`Color_Red`，`Color_Green`。或者：`Planet_Earth`，`Planet_Mars`）。对于枚举成员总是被冠以枚举字前缀的编程语言（例如 Java），可以将上述名字简化为：`Color.Red` 和 `Planet.Earth`。或者：

- 为常量命名（**Naming Constants**）

  常量命名应当根据该常量所表示的含义进行命名，而不是其数值。例如：`CYCLES_NEEDED` 比 `FIVE` 要好。

11.3 命名规则的力量（The Power of Naming Conventions）

有效的命名规则是最强大的工具之一。

- 为什么要有规则（**Why Have Conventions**）

  命名规则的存在为代码增加了结构，减少了我们需要考虑的事情。命名规则的好处：

  - 要求我们按规矩行事，集中精力关注代码更重要的特征。
  - 有助于在项目之间传递知识，名字的相似性将能帮助我们更容易理解不熟悉的变量。
  - 一致风格的代码有助于在新项目中更快速地学习。
  - 有助于减少名字增生（name proliferation），即给同一个对象起两个不同的名字。
  - 弥补编程语言的不足之处，用规则来效仿具名常量和枚举类型。规则可以根据局部数据、类数据以及全局数据的不同而有所差别，并且可以包含编译器不直接提供的类型信息。
  - 强调相关变量之间的关系。

- 何时采用命名规则（**When You Should Have a Naming Conventions**）

  在下列情况下命名规则很有价值：

  - 多个程序员合作开发一个项目。
  - 把程序交给另一位程序员来修改和维护的时候。
  - 当其他程序员评估你所写的代码的时候。
  - 程序规模太大，以至于无法再脑海里同时了解事情的全貌，而必须分而治之的时候。
  - 当程序生命期足够长，长到可能会把它搁置几个星期或几个月之后又重新启动有关程序的工作时。
  - 当一个项目中存在一些不常见的术语，并且你希望在编写代码阶段使用标准的术语或者缩写的时候。

- 正式程度（**Degrees of Formality**）

  对于微小的，用完即弃的项目而言，严格的规则可能没有必要。对于多人协作的大型项目，在任何阶段，正式规则都是成为提高可读性的必不可少的辅助手段。

11.4 非正式命名规则（Informal Naming Conventions）

大多数项目采用类似于本节所讲的相对非正式的指导规则。

- 与语言无关的命名规则的知道原则（**Guidelines for Language-Independent Conventions**）
  - 区分变量名和子程序名字。本书所采用的命名规则要求变量名和对象名以小写字母开始，例如： `variableName` ；而子程序名字以大写字母开始，例如： `RoutineName()` 。
  - 区分类和对象。类名字与对象名字（或者类型与该类型的变量）。一些标准的方案如下：
    - 通过大写字母开头区分类型和变量（ `Widget` 和 `widget` ）。
    - 通过全部大写区分类型和变量（ `WIDGET` 和 `widget` ）。
    - 通过给类型加 `t_` 前缀区分类型和变量（ `t_Widget` 和 `Widget` ）。
    - 通过给变量加 `a` 前缀区分类型和变量（ `Widget` 和 `aWidget` ）。
    - 通过对变量采用更明确的名字区分类型和变量（ `Widget` 和 `employeeWidget` ）。
  - 标识全局变量。例如：在所有的全局变量名之前加上 `g_` 前缀。
  - 标识成员变量。根据名字识别出变量是类的数据成员，即明确表示该变量既不是局部变量，也不是全局变量。例如：用 `m_` 前缀来标识类的成员变量。
  - 标识类型声明。为类型建立命名规则有两个好处：能够明确表明一个名字是类型名；能够避免类型名与变量名冲突。增加前缀或者后缀是不错的方法，例如：为类型名增加 `t_` 前缀，如 `t_Color` 和 `t_Menu` 。
  - 标识具名常量。对具名常量加以标识，可以明确在为变量赋值时使用的是另一个变量的值还是一个常量的值。给常量命名的方法之一是给常量名增加 `c_` 前缀。或者使用下划线分割的前部大写单词（例如： `LINES_PER_PAGE_MAX` ）。
  - 标识枚举类型的元素。标准方法为：全部用大写；或者为类型名增加 `e_` 或者 `E_` 前缀，同时为该类型的成员名增加基于特定类型的前缀（例如： `Color_` 或者 `Planet_` ）。
  - 在不能保证输入参数只读的语言里标识只读参数。例如对于对象内容可被修改的程序中，为输入参数增加一个 `const` 或者 `final` 的前缀，就知道该变量是不应该被修改的。
  - 格式化命名以提高可读性。用大小写和分隔符来分割单词是两种可以用来提高可读性的分隔方法。尽量不要混用方

法。

- 与语言相关的命名规则的指导原则（**Guidelines for Language-Specific Conventions**）

  应当遵循所使用语言的命名规则。

  - C 的命名规则（C Conventions）（UNIX / LINUX 风格的 C 编程规则）
    - `c / ch` 是字符变量。
    - `i / j` 是整数下标。
    - `n` 表示事物的数量。
    - `p` 是指针。
    - `s` 是字符串。
    - 预处理宏全部大写（ `ALL_CAPS` ）。这通常也包括 `typedef` 。
    - 变量名和子程序名全部小写（ `all_lowercase` ）。
    - 下划线（ `_` ）用作分隔符（例如： `letters_in_lowercase` ）。
  - C 命名规则（Microsoft Windows 平台）
    - 使用匈牙利命名法。
    - 在变量中混合使用大小写。
  - C 命名规则（Macintosh 平台）
    - 在子程序的名字中混合使用大小写。
  - C++ 的命名规则（C++ Conventions）
    - `i / j` 是整数下标。
    - `p` 是指针。
    - 常量、 `typedef` 和预处理宏全部大写（ `ALL_CAPS` ）。
    - 类和其他类型的名字混合大小写（ `MixedUpperAndLowerCase()` ）。
    - 变量名和函数名中的第一个单词小写，后续每个单词的首字母大写（例如： `variableOrRoutineName` ）。
    - 不把下划线用作名字中的分隔符，除非用于全部大写的名字以及特定的前缀中（如用于标识全局变量的前缀）。
  - Java 的命名规则（Java Considerations）
    - `i / j` 是整数下标。
    - 常量全部大写（ `ALL_CAPS` ）并用下划线分割。
    - 类名和接口名中每一个单词的首字母均大写，包括第一个单词（例如： `ClassOrInterfaceName` ）。
    - 变量名和方法名中第一个单词的首字母小写，后续单词的首字母大小（例如： `variableOrRoutineName` ）。
    - 除用于全部大写的名字之外，不使用下划线作为名字中的分隔符。
    - 访问器子程序使用 `get` 和 `set` 前缀。

- 混合语言编程的注意事项（**Mixed-Language Programming Considerations**）

  在混合语言环境中编程时，可以对命名规则做出优化以提高整体的一致性和可读性（即使这样会使规则与某个语言自身的规则冲突）。例如本书中的子程序名首字母大写遵循了 C++ 的规则，但是这与 Java 中所有方法名以小写字母开始冲突。

- 命名规则示例（**Sample Naming Conventions**）

  C++ 和 Java 的命名规则示例：

```
  +---------------------+-------------------------------------------------------------------------------
----------------------+
  | Entity              | Description
|
  +---------------------+-------------------------------------------------------------------------------
----------------------+
  | ClassName           | 类名混合使用大小写，首字母大写。
  | TypeName            | 类型定义，包括枚举类型和 typedef，混合使用大小写，首字母大写。
  | EnumeratedTypes     | 除遵循上述规则之外，枚举类型总以复数形式表示。
  | localVariable       | 局部变量混合使用大小写，首字母小写。其名字应该与底层数据类型无关，而且应该反映该变量所
代表的事物。
```

```
  | routineParameter    | 子程序参数的格式与局部变量相同。
  | RoutineName()       | 子程序名混合使用大小写。
  | m_ClassVariable     | 对类的多个子程序可见（且只对该类可见）的成员变量名用 m_ 前缀。
  | g_GlobalVariable    | 全局变量名用 g_ 前缀。
  | CONSTANT            | 具名常量全部大写。
  | MACRO               | 宏全部大写。
  | Base_EnumeratedType | 枚举类型名用能够反映其基础类型的、单数形式的前缀 - 例如：Color_Red, Color_Blue。
  +---------------------+-------------------------------------------------------------------------
----------------------+
```

C 的命名规则示例：

```
  +-------------------------+-------------------------------------------------------------------------
-------------------------------------------------+
  | Entity                  | Description
|
  +-------------------------+-------------------------------------------------------------------------
-------------------------------------------------+
  | TypeName                | 类型名混合使用大小写，首字母大写。
  | GlobalRoutineName()     | 公用子程序名混合使用大小写。
  | f_FileRoutineName()     | 单一模块（文件）私用的子程序名用 f_ 前缀。
  | LocalVariable           | 局部变量呼和使用大小写。其名字应该与底层数据类型无关，而且应该反映变量所代表的事
物。
  | RoutineParameter        | 子程序参数的格式与局部变量相同。
  | f_FileStaticVariable    | 模块（文件）变量名用 f_ 前缀。
  | G_GLOBAL_GlobalVariable | 全局变量名以 G_ 前缀和一个能反映定义改变量的模块（文件）的、全部大写的名字开始 -
例如，G_SCREEN_Dimensions。
  | LOCAL_CONSTANT          | 单一子程序或者模块（文件）私用的具名常量全部大写 - 例如，ROWS_MAX。
  | G_GLOBALCONSTANT        | 全局具名常量名全部大写，并且以 G_ 前缀和一个能反映定义该具名常量的模块（文件）的、
全部大写的名字开始，如 G_SCREEN_ROWS_MAX。
  | LOCALMACRO()            | 单一子程序或者模块（文件）私用的宏定义全部大写。
  | G_GLOBAL_MACRO()        | 全局宏定义全部大写，并且以 G_ 前缀和一个能反映定义该宏的模块（文件）的全部大写名字
开始 - 例如，G_SCREEN_LOCATION()。
  +-------------------------+-------------------------------------------------------------------------
-------------------------------------------------+
```

变量名包含了以下三类信息：

- 变量的内容（它代表什么）
- 数据的种类（具名常量，简单变量，用户自定义类型或者类）
- 变量的作用域（私用的，类的，包的或者全局的作用域）

11.5 标准前缀（Standardized Prefixes）

对具有通用含义的前缀标准化，为数据命名提供了一种简洁、一致并且可读性好的方法。有关标准前缀最广为人知的方案是匈牙利命名法。标准化的前缀由两部分组成：用户自定义类型（UDT）的缩写和语意前缀。

- 用户自定义类型缩写（**User-Defined Type Abbreviations**）

  UDT 缩写可以标识被命名对象或变量的数据类型。UDT 用很短的编码描述，这些编码有助于用户理解其所代表的实体（例如： `wn` 代表窗体 window）。

- 语义前缀（**Semantic Prefixes**）

  语义前缀比 UDT 更进一步，描述了变量或者对象是如何使用的。语义前缀在某种程度上对于不同的项目是标准的（例如： `g` 代表全局变量， `i` 代表下标， `p` 代表指针， `min` ， `max` ， `first` ， `last` 等），而 UDT 会根据项目的不同而不同。

  语义前缀可以全用小写，也可以混合使用大小写，还可以根据需要与 UDT 和其他的语义前缀结合使用。

- 标准前缀的优点（**Advantages of Standardized Prefixes**）

  除了命名规则所提供的一般意义上的优点外，标准前缀的另外一些好处：

  - 能够更为精确地描述一些含义比较模糊的名字： `min` 、 `first` 、'last' 和 `max` 之间的严格区别就显得格外有用。
  - 标准化的前缀使名字变得更加紧凑。但是为了提高可读性，不能忽略给变量起更有意义的名字（例如： `ipa` 虽然已经明确地表示这是一个段落数组的下标了，但是 `ipaActiveDocument` 这样的名字更有意义）。
  - 在编译器不能检查所使用的抽象数据类型的时候，标准前缀能帮助我们准确地对类型做出判断： `paReformat =`

`docReformat` 很可能不对，因为 `pa` 和 `doc` 是不同的 UDT。

11.6 创建具备可读性的短名字（Creating Short Names That Are Readable）

对于现代语言如 C++，Java 等，几乎没有任何理由去缩短具有丰富含义的名字。如果环境真的要求创建简短的名字，有些缩短名字的方法要好于其他方法：消除冗余的单词、使用简短的同义词以及使用诸多缩写策略中的任意一种来创建更好的短变量名。

- 缩写的一般指导性原则（**General Abbreviation Guidelines**）

  以下为几项用于创建缩写的指导原则。其中一些彼此冲突，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` 变成 `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个和最后一个字母。
  - 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。
  - 保留每个音节中最引人注意的发声。
  - 确保不要改变变量的含义。
  - 反复使用上述技术，知道你把每个变量民的长度缩减到了 `8` 到 `20` 个字符，或者达到你所使用的编程语言对变量名的限制字符数。

- 语音缩写（**Phonetic Abbreviations**）

  有些人提倡基于单词的发音而不是拼写来创建缩写。这不是提倡的做法（例如： `highlight` 变成了 `hilite` ， `execute` 变成了 `xqt` ）。

- 有关缩写的评论（**Comments on Abbreviations**）

  以下是一些能够用来避免犯错的规则：

  - 不要用从每个单词中删除一个字符的方式来缩写。要么删除不止一个字符，要么把单词拼写完整。节省一个字符带来的便利很难抵消由此而造成的可读性损失（例如： `jun` 和 `jul` ）。
  - 缩写要一致。例如：要么全用 `Num` ，要么全用 `No` ，不要两个都用。也不要 `Number` 和 `Num` 混用。
  - 创建你能读出来的名字。例如：用 `xPos` 而不用 `xPstn` ，用 `needsCompu` 而不用 `ndsCmptg` 。
  - 避免使用容易看错或者读错的字符组合。例如： `ENDB` 比 `BEND` 要好，但是 `BEnd` 或者 `B_END` 更好一些。
  - 使用辞典来解决命名冲突。短名字可能会造成命名冲突，避免这一情况的一种简单做法是使用含义相同的不同单词，例如：使用 `dismissed` 来代替 `fired` ，用 `complete revenue disbursal` 来代替 `full revenue disbursal` ，这样缩写就分别变成了 `dsm` 和 `crd` ，而不是冲突的 `frd` 。
  - 在代码里用缩写对照表解释极短的名字的含义。
  - 在一份项目级的“标准缩写”文档中说明所有的缩写。
  - 记住，名字对代码读者的意义比作者更重要。

11.7 应该避免的名字（Kinds of Names to Avoid）

有关应当避免的变量名的指导原则：

- 避免使用令人误解的名字或缩写。
- 避免使用具有相似含义的名字。如果能够交换两个变量的名字而不会妨碍对程序的理解，那么就需要为这两个变量重新命名了（例如： `input` 和 `inputValue` ）。
- 避免使用具有不同含义但却有相似名字的变量。例如： `clientsRecs` 和 `clientsReps` 。应当采用至少有两个字母不同的名字（例如： `clientsRecords` 和 `clientsReports` 就好很多）。
- 避免使用发音相近的名字。例如： `wrap` 和 `rap` 。
- 避免在名字中使用数字。如果名字中的数字真的非常重要，就使用数组来代替一组单个的变量。如果数组不合适，那么数字（例如： `file1` ， `total1` ）就更不合适。特殊情况下也可以使用数字（例如：203 国道）。

- 避免使用英语中常常拼错的单词。例如： `acsend` , `prefered` 。
- 不要仅靠大小写来区分变量名。
- 避免使用多种自然语言。也应该避免使用多种英语变体（例如： `color` 和 `colour` ）。
- 避免使用标准类型、变量和子程序的名字。
- 不要使用与变量含义完全无关的名字。例如： `Tom` 等人名或啤酒名等。
- 避免在名字中包含易混淆的字符。例如： `1`（数字） 、 `l`（小写字母 `l`） 和 `I`（大写字母 `I`） ， `0`（零） 和 `O`（大写字母 `O`） ， `2` 和 `Z` ， `S` 和 `5` ， `G` 和 `6` 。

核对表：变量命名（**CHECKLIST: Naming Variables**）

命名的一般注意事项：

- 名字完整并准确地表达了变量所代表偶读含义吗？
- 名字反映了现实世界的问题而不是编程语言方案吗？
- 名字足够长，可以让你无须苦苦思索吗？
- 如果有计算限定符，它被放在名字的最后吗？
- 名字中用 `Count` 或者 `Index` 来代替 `Num` 了吗？

为特定类型的数据命名：

- 循环下标的名字有意义吗（如果循环的长度超出了一两行代码或者出现了嵌套循环，那么就应该是 `i` ， `j` 或者 `k` 以外的其他名字）？
- 所有的"临时"变量都重新命以更有意义的名字了吗？
- 当布尔变量的值为真时，变量能够准确表达其含义吗？
- 枚举类型的名字中含有能够表示其类别的前缀或后缀了吗？例如，把 `Color_` 用于 `Color_Red` ， `Color_Green` ， `Color_Blue` 等了吗？
- 具名常量是根据它所代表的抽象实体而不是它所代表的数字来命名的吗？

命名规则：

- 规则能够区分局部数据、类的数据和全局数据吗？
- 规则能够区分类型名，具名常量、枚举类型和变量名吗？
- 规则能够在编译器不强制检测只读参数的语言里表示出子程序中的输入参数吗？
- 规则尽可能地与语言的标准兼容吗？
- 名字为了可读性而加以格式化吗？

短名字：

- 代码用了长名字吗（除非有必要使用短名字）？
- 是否避免只为了省一个字符而缩写名字的情况？
- 所有单词的缩写方式都一致吗？
- 名字能够读出来吗？
- 避免使用容易被看错或者读错的名字吗？
- 在缩写对照表里短命子做出说明吗？

常见命名问题：你是否做到了避免使用 ...

- 容易让人误解的名字吗？
- 有相近含义的名字吗？
- 只有一两个字符不同的名字吗？
- 发音相近的名字吗？
- 包含数字的名字吗？
- 为了缩短而故意拼错的名字吗？
- 英语中经常拼错的名字吗？
- 与标准库子程序名或者预定义变量名冲突的名字吗？
- 过于随意的名字吗？
- 含有难读的字符的名字吗？