第 1 章

将问题转变成代码

如果你才接触编程，可能想知道有经验的开发者是如何看问题以及如何将其转变为可运行的代码的。其实，编写代码只是这个过程中很小的一部分。在解决问题之前，首先要将其分解。如果你观察过有经验的开发者，可能会觉得他们看上去就是打开代码编辑器，噼里啪啦敲出了一个解决方案。但是多年以来，他们分解的问题就算没有几千个，也有几百个了，他们能看出一些模式。而新手可能并不知道怎么做。所以，本章将研究一种分解问题并将其转变为代码的方法。你可以使用该方法解决本书其余部分的问题。

理解问题

要想知道应该做什么，最好的一个方法是把它写下来。如果我告诉你，我想要一个小费计算程序，是不是凭这个信息你就能开工写一个了？可能还不够。你可能还必须向我提一些问题。这通常称作收集需求，不过我喜欢把它看作找出程序应该提供的特性。

先考虑一些可以向我提出的问题，以便通过这些问题更好地理解我想要的是什么。要构建这个应用，你需要了解什么呢？

想到一些问题了吗？很好。下面是一些你可能想问的问题。

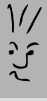
- 你想使用什么样的公式？可以解释一下如何计算小费吗？
- 小费的比例是多少？是固定的 15%，还是用户可以修改？
- 程序启动后，屏幕上应该显示什么？
- 程序应该如何显示其输出？你想看小费和总额，还是只想看总额？

一旦得到这些问题的答案，就可以尝试写一下问题描述，精确地解释要构建的东西。下面是欲构建程序的问题描述：

> 创建一个简单的小费计算程序。该程序应该提示用户输入账单金额和小费比例。该程序必须计算出小费，并显示小费和总金额。

示例输出：

```
What is the bill? $200
What is the tip percentage? 15
The tip is $30.00
The total is $230.00
```



小乔爱问：

如何处理复杂的程序？

将较大的程序分解成更容易管理的、较小的特性。这么做的话，因为每个特性可以具体化，所以成功的概率更大。已有的大部分复杂应用都是由很多较小的、一起工作的程序组成的。Linux 上的命令行工具就是这么工作的；一个程序的输出可以作为另一个程序的输入。

如果你这就准备打开文本编辑器敲代码，还是有点太着急了。听我说，

如果不花时间仔细设计，或许也能开发出能够工作的程序，但是质量并不高。而且不幸的是，很容易出现一些不好的事情。比如，你写出的程序没有测试、计划，也没有文档，而你的老板看到这个程序后，以为完事了，让你发布。现在产品中有未经测试、没有计划的代码，以后可能还会有人让你做些修改。设计差劲的代码很难维护和扩展。所以就让我们以小费计算程序为例，从头到尾过一遍这个能够帮你理解应该构建什么的简单过程。

发现输入、处理和输出

不管是小费计算程序这样的简单应用，还是 Facebook 那样的复杂应用，每个程序都有输入、处理和输出。事实上，大型应用就是由一系列较小的、相互通信的程序组成的。其中，一个程序的输出会成为另一个程序的输入。

不管程序是大是小，如果能花点时间清晰地阐明输入、处理和输出各是什么，就能确保其工作地很好。如果有比较清晰的问题描述，则有个简单的方法，就是看一下问题描述中的名词和动词。名词最终会成为输入和输出，动词会成为处理。看一下小费计算程序的问题描述：

> 创建一个简单的小费计算程序。该程序应该提示用户输入账单金额和小费比例。该程序必须计算出小费，并显示小费和总金额。

首先，找找名词。如果你喜欢，可以拿笔圈起来，也可以列个表。下面是我列的表：

- 账单金额
- 小费比例

- 小费
- 总金额

再来看看动词有哪些呢？

- 提示
- 计算
- 显示

所以我们知道了，必须提示用户输入，做些计算，然后显示输出。通过考虑名词和动词，可以了解要求我们做的是什么。

当然，问题描述未必总是这么清晰。比如，问题描述说需要计算小费，但又说需要显示小费和总金额。这意味着需要把小费和原始的账单金额相加，得到那个输出。这是构建软件的挑战之一。它未必总会被讲清楚。但随着经验越来越丰富，你将能弥合这种差异，领会字里行间的言外之意。

借助一点侦查工作，我们确定了这个程序的输入、处理和输出，如下所示。

- 输入：账单金额，小费比例
- 处理：计算小费
- 输出：小费、总金额

我们是不是已经为编写代码做好准备了呢？还没有。

用测试驱动设计

设计和开发软件的最佳方式之一，是从一开始就思考想要的结果。很

多职业软件开发人员会采用一种叫作测试驱动开发（Test-Driven Development，TDD）的正规过程做到这一点。在 TDD 中，你会编写很多代码测试程序的输出，或者是测试组成大型程序的单个程序的输出。随着开发的进行，这个测试过程会指导你走向好的设计，并帮助你思考程序可能存在的问题。

TDD 确实需要一些所用语言相关的知识，还需要一点初学者尚不具备的经验。

然而，TDD 的本质是提前思考程序的预期结果是什么，然后为此努力。如果在编写代码之前做到这一点，我们的思维就能超越最初的需求。即使我们并不能很舒服地按照 TDD 的正规流程做下来，只是创建一些简单的测试计划，也有不少好处。测试计划会列出程序的输入和预期结果。

测试计划看上去就像这样：

```
输入：
预期结果：
实际结果：
```

列出程序的输入，然后写出应该得到的输出。运行程序，比较预期结果与程序给出的实际结果。

让我们通过思考小费计算程序来实践一下。我们怎么知道程序的输出应该是什么呢？我们怎么知道计算是否正确呢？

通过一些测试计划来定义我们希望程序如何工作。先写一个非常简单的：

```
输入：
  账单金额：10
```

```
   小费比例：15
预期结果：
   小费：$1.50
   总金额：$11.50
```

这个测试计划告诉我们两件事。首先，它告诉我们，程序需要两个输入：账单金额 10 和小费比例 15。因此，在做数学运算时，需要将小费比例从整数转换成小数。它还告诉我们，我们将以货币形式打印小费和总额。所以我们知道了，程序中最好做些转换。

一个测试并不够。如果使用 11.25 作为输入呢？测试计划中，输出应该是什么？试试看。填充下面的测试计划：

```
输入：
   账单金额：11.25
   小费比例：15
预期结果：
   小费：???
   总金额：???
```

假设你用计算器算出了小费，得到的结果或许会是 1.6875。

但是这现实吗？可能并不现实。我们可能会向上取整到最接近的分上。所以测试计划将是这样的：

```
输入：
   账单金额：11.25
   小费比例：15
预期结果：
   小费：$1.69
   总金额：$12.94
```

我们刚刚使用一个测试来设计了程序的功能；我们确定了程序需要对答案中的分做向上取整。

在你做本书中的练习时，花些时间，为每个程序设计至少 4 个测试计划，并尝试尽可能多地考虑别人可能会使程序出错的场景。随着接触更复杂的问题，你可能需要更多的测试计划。

如果你是想从 TDD 开始的有经验的开发人员，则应该使用本书中的练习来熟悉你喜爱的编程语言所提供的库和工具。可以在维基百科[①]上找到很多编程语言中有的测试框架。可以阅读 Kent Beck 的《测试驱动开发》来获得更多信息，了解如何设计带测试的代码，还可以调研更多与 TDD 相关的更特定于语言的资源。

现在我们对小费计算程序应该具有的特性有了更清晰的认识，可以开始组合程序的算法了。

用伪代码编写算法

*算法*就是需要执行的一组一步一步的操作。如果拿到一个算法，编写代码执行这些操作，最后会得到一个计算机程序。

如果才学编程，尚未完全熟悉一门编程语言的语法，则应该考虑用*伪代码*把算法写出来。伪代码语法和英语类似，方便我们思考逻辑，还不用担心纸的问题。伪代码并不是只有初学者才用得着；在和同事一起解决问题时，甚至是一个人解决问题时，有经验的程序员偶尔也会在白板上写一些伪代码。

编写伪代码并没有所谓“正确的方式”，尽管有些应用较广的术语。可以使用 `Initialize` 表示设置初始值，使用 `Prompt` 表示提示用户

① https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

输入，以及使用 Display 表示将内容显示到屏幕上。

小费计算程序用伪代码可以这样写：

```
TipCalculator
  Initialize billAmount to 0
  Initialize tip to 0
  Initialize tipRate to 0
  Initialize total to 0

  Prompt for billAmount with "What is the bill amount?"
  Prompt for tipRate with "What is the tip rate?"

  convert billAmount to a number
  convert tipRate to a number

  tip = billAmount * (tipRate / 100)
  round tip up to nearest cent
  total = billAmount + tip

  Display "Tip: $" + tip
  Display "Total: $" + total
End
```

从这段伪代码中可以大致看出程序的算法。必须设置一些变量，根据输入作出决策，进行转换，然后将输出显示到屏幕上。我建议在伪代码中，包含像变量的名字和要在屏幕上显示的文本等细节，因为这可以帮助我们更清晰地思考程序的最终结果。

这是编写程序的最佳方式吗？或许不是。但这并不是关键所在。通过编写伪代码，我们创建出了一些可以演示给其他开发者以获得反馈的东西。把它们扔到一块花不了太多时间。

最好的情况是，我们可以将其用作一个蓝图，可以用任何编程语言编码实现。请注意，伪代码中并没有假定最终会使用何种编程语言，但

是确实提供了一些指导，像变量名是什么，显示给最终用户的输出是什么，等等。

一旦有了程序的一个初始版本，并让它跑起来，就可以开始修改代码来加以改进了。比如，可能是将程序划分成函数，或者是将数学转换放到一起做，而不是分开进行。就是把伪代码看作一种计划工具。

编写代码

现在轮到你了。利用所学的知识，是不是可以编写出这个程序的代码了？试试吧。在做的时候要记住下面这些约束。

约束

- 小费比例应输入百分比的数字部分。比如，15%的小费比例，应该输入 15，而不是 0.15。应该由程序来处理除法。
- 不足一分的，向上取整。

如果不知道如何实施这些约束，可以暂不考虑，先编写程序，以后再回来看。这些习题的目的就是练习和提高。

如果现在感觉这个程序挑战很大，可以先跳过，先做做本书中那些更容易的，然后再回到这个问题。

挑战

在编写出一个基本的版本后，可以再试试解决下面这些挑战。

- 对于账单金额和小费比例，确保用户只能输入数值。如果用户输入的不是数值，则显示相应的提示信息并退出程序。下面是测试计划的一个例子：

```
输入：
    账单金额：abcd
    小费比例：15
预期结果：请输入一个合法的账单金额数值。
```

- 不再是显示错误消息并退出，而是在用户输入合法的值之前，一直提示输入。
- 不允许用户输入负数。
- 将程序分解为做计算的函数。
- 使用图形用户界面（GUI）实现这个程序，当有任何值改变时，自动在界面中更新这些值。
- 不再是让用户以百分比形式输入小费比例，而是让用户通过拖动一个滑动条表示对服务员的满意度，区间是 5%~20%。

前进！

尝试使用该策略解决本书中的每个问题，从而使收益最大化。发现输入、处理和输出。开发些测试计划，想出伪代码，编写程序，然后接受每个程序后面的各种挑战。或者是选择自己的方向。再或者是用尽可能多的语言编写这个程序。

但最重要的是，玩得开心，享受学习。