第 3 章

条件和循环

经过前两章的学习后，我相信你会很愿意用 Julia 创建一些真实的应用程序。这些应用程序的工作将由你的响应决定，有时还决定于你希望它们如何工作。然而，在你可以做到这一点之前，你首先必须学习一些关于编程的基本知识：如何让计算机做出决定以及如何遍历数据。

在本章中，你将学习：

- 什么是条件（condition）；
- 条件操作符是什么；
- 计算机如何使用 if/else-if/else 语句检查和断言条件进行决策；
- 什么是迭代（iteration）；
- 如何使用 for 循环迭代；
- 如何使用 while 循环迭代。

3.1 什么是条件

在我们关心的上下文中，条件是“一种情况或一种结果”。当与 if 一起使用时，条件将导致真或假（是或否），程序控制可以转向一条（第一条）或另一条（第二条）路径。

这是使计算机能够运行的最基本和最关键的操作，你使用计算机所做的几乎所有操作都有与之相关的某种条件。例如：

① 当你在 Chrome 浏览器中打开新选项卡时，浏览器必须检查是否已打开过多的选项卡。

② 当你注册在线服务时，表单需要根据你的出生日期确保你的年龄足够大。

③ 当你使用键盘在文本编辑器上输入时，编辑器需要检查 Caps Lock 键是否打开或是否按下 Shift 键。

④ 当你问 Siri“我今天需要穿雨衣吗?”，手机需要查询天气是否会下雨。

以上只是几个简单的例子，在日常生活中，几乎所有不同场景中使用的应用程序都会使用条件。

下面你将学习如何在 Julia 语言中实现这些条件。

3.2 条件操作符是什么

假设我们要构建一个应用程序，用来检查某个用户是否超过某一年龄阈值，以便让该用户享受相关服务。继续操作并打开名为 old_enough.jl 的新文件，输入以下代码：

```
1   minimum_age = 18
2   print("Enter your age: ")
3   user_age = parse(Int64, readline())
```

我相信你还记得第 2 章中的这段代码做了以下三件事：

① 定义一个 minimum_age 变量，赋值为 18；

② 打印提示信息，以便用户提供其年龄信息；

③ 从用户处读取输入，将其从字符串转换为整数形式，并存储在 user_age 中。

一旦获取了 user_age 变量，就需要确定该 user_age 变量的值是否足够大，使用户能够进入服务行列。我们可以使用一个条件运算符实现这一点。

在继续讨论条件之前，我们需要先了解一些将要使用的简单术语。什么是操作符呢？让我们举个例子：

```
user_age > minimum_age
```

这行代码读作 user_age 大于 minimum_age，这里的大于(>)符号是一个操作符，因为它分别对写在其左右两侧的 user_age 和 minimum_age 进行操作。此外，user_age 和 minimum_age 也被称为操作数。这样写，表示这是一个评估两种可能结果之一(true 或 false)的条件，取决于存储在左右变量中的值。我们将在创建的应用程序中学习并使用许多这样的操作符。

正如我们刚刚学习过的，加法(+)符号需要读取两个数并返回它们的和，星号(*)符号需要读取两个数并返回它们的乘积。相比之下，条件操作符需要接收一个或两个值，并返回一个布尔(true 或 false)值。

Julia 中有很多不同的条件操作符，以下是其中的一些。

操作符	叫法	描　　述
>	大于	需要两个值，并检查左边的值是否大于右边的值
<	小于	需要两个值，并检查左边的值是否小于右边的值
==	等于	需要两个相同类型的值，并检查它们是否都表示相同的值
!=	不等于	需要两个值，并检查它们是否都表示不同的值
<=	小于或等于	需要两个值，并检查左边的值是否小于或等于右边的值
>=	大于或等于	需要两个值，并检查左边的值是否大于或等于右边的值
!	布尔逆	这个运算符有点不同，它只需要右边的一个值，将返回它所传递的值的逆。例如，如果传递 false，则将返回 true；如果传递 true，则返回 false

备注：==操作符比较特殊，并且不同于=操作符。=操作符用来为一个变量赋值，==操作符用来比较两个变量，结果返回 true 或 false。

在这个例子中，我们希望确保用户的年龄大于或等于我们定义的最小年龄。要想执行此操作并打印出结果，需要添加以下代码：

```
4  println(user_age >= minimum_age)
```

好了！如果只使用 4 行代码运行应用程序，那么你会得到一个提示，要求输入你的年龄。如果输入的值小于 18，则会显示 false；否则会显示 true。

但这并不是很有趣，也没有做太多的事情，所以让我们带着它进入下一级。如果条件操作符返回 true，则要执行一些代码；否则要执行其他代码，这就是 if 语句出现的地方！

3.3　计算机如何使用 if/else-if/else 语句进行决策

使用 if 语句可以根据条件为 true 还是 false 控制代码流。让我们先来看一个例子。

首先，继续从 old_enough.jl 文件中删除最后一行代码，然后输入代码

```
4  if user_age >= minimum_age
5      println("You're old enough. Welcome!")
6  end
```

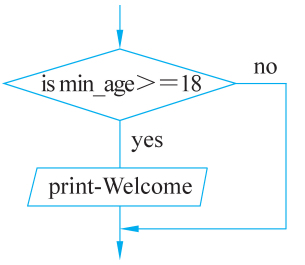
让我们来解析一下这三行代码。

① 第 4 行告诉 Julia 我们正在开始一个 if 语句，并提供了一个需要检查的条件。

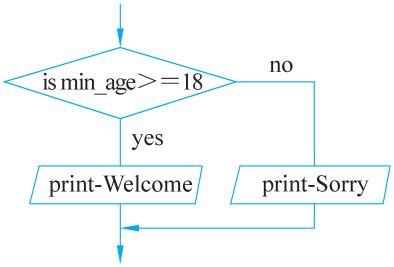
② 如果第 4 行中的条件为 true，则执行第 5 行。

③ 第 6 行告诉 Julia，if 语句的代码块已经结束。如果第 1 行（行 4）中的条件为 true，那么在这一行之前和第一行（行 4）之后的所有内容都将被执行，它可以是这两行之间任意数量的代码行。这一部分代码被称为一个代码块。

现在，如果要运行应用程序，你会看到提示，但还有一些问题。如果你提供的年龄足够大，即 18 岁或以上，则它确实会通过 println 函数在下一行打印消息。然而，如果不是，那么什么也不会发生。程序并没有说你的年龄还不够大，只是什么也没做。右面这张流程图将会让你看得更清楚。



但是，你想要的流程应该是这样的：



这种情况不会发生，因为我们还没有告诉 if 语句当判断结果为 false 时要执行什么代码。

```
if user_age < minimum_age
    println("You're too young — sorry!")
end
```

这样就可以正常工作了，但这还不是最理想的，你明明已经检查过一次条件了，为什么还要再检查一次呢？相反，你可以使用 else 子句，它是一个在条件非 true 时提供代码的子句。

替换已有的 if 语句：

```
.
4   if (user_age >= minimum_age)
5       println("You're old enough. Welcome!")
6   else
7       println("You're too young — sorry!")
8   end
```

这样就足够了！如果运行应用程序，那么现在应该可以看到输入年龄大于和小于阈值时的输出，并告诉它们是否符合条件。

但是你还可以做更多的事情。例如，如果用户的年龄超过了阈值，你想欢迎他们，让他们进入；如果他们的年龄正好是阈值年龄，则要发出警告；如果他们的年龄小于阈值年龄，就不允许他们进入，等等。当然，你可以编写以下这样的代码：

```
.
4  if user_age > minimum_age
5      println("You're old enough. Welcome!")
6  else
7      if user_age == minimum_age
8          println("You're old enough, but be careful.")
9      else
10         println("You're too young — sorry!")
11     end
12 end
```

事实上，我建议你尝试一下，以便通过输入不同的年龄了解程序是如何工作的。但是，还有一种更有效的方法——if/else-if/else 语句。

让我们来看一个例子，然后我会解释一下它是如何工作的。

```
.
4  if user_age > minimum_age
5      println("You're old enough. Welcome!")
6  elseif user_age == minimum_age
7      println("You're old enough, but be careful.")
8  else
9      println("You're too young — sorry!")
10 end
```

让我们逐行来看一下，从第 4 行开始。

① 第 4 行将检查用户的年龄是否大于最小年龄。

② 如果第 4 行的条件为 true，则运行第 5 行并打印消息“You’re old enough. Welcome!”。

③ 如果第 4 行的条件为 false，则第 6 行将检查用户的年龄是否等于最小年龄。

④ 如果第 6 行的条件为 true，而第 4 行的条件为 false，则运行第 7 行并打印消息“You’re old enough, but be careful!”。

⑤ 第 8 行准备让 Julia 看看即将到来的代码。

⑥ 如果第 4 行和第 6 行中的条件都为 false，则运行第 9 行。

⑦ 第 9 行打印消息“You're too young—sorry!”。

⑧ 第 10 行告诉 Julia，你在第 4 行开始的 if 语句的代码块已经结束。

⑨ 事实上，你也可以编写多个 else-if 语句，例如：

```
.
4  if (user_age > minimum_age + 1)
5      println("You're old enough. Welcome!")
6  elseif (user_age == minimum_age + 1)
7      println("You're old enough, and I'm sure you'll
       ⇨  be fine.")
8  elseif (user_age == minimum_age)
9      println("You're old enough, but be careful!")
10 else
11     println("You're too young — sorry!")
12 end
```

将对每个条件进行判断，直到其中一个条件为 true，然后运行其相应的代码块。如果没有判断为 true 的条件，则运行与 else 语句对应的代码块。

现在，你已经了解了条件以及 if 语句是如何工作的了。

练习 1

现在是时候练习一下如何创建以下应用程序了。

1. 提示用户输入一个整数，并通知用户输入的数字为偶数或奇数。

2. 提示用户输入一个整数，并通知用户输入的数字是否可以被 7 整除。

3. 提示用户输入两个整数作为平行四边形的底和高，并计算出面积。

3.4　什么是迭代

正如前面提到的，条件是编程的重要组成部分，但仅仅如此还有点不完整。为了让计算机真正发挥作用，它们还需要支持迭代，即多次执行任务。

事实上，如果你仔细想想就会明白，这才是我们使用计算机的初衷，因为它们可以数百万次地做同一件事情，不会觉得“无聊”或需要“休息”，这可是我们人类做不到的！

Julia 中有两种迭代的方法——for 循环和 while 循环。前者使用得更广泛，而后者则更灵活一些。

3.5　如何使用 for 循环迭代

实际上，for 循环非常简单，请打开一个名为 countUpTo.jl 的新文件，然后输入代码

```
for iteration_number in 1:5
    println(iteration_number)
end
```

我将会解释这段代码的作用，但请先运行一下代码，你应该会看到以下输出：

```
1
2
3
4
5
```

这段代码的本意是告诉 Julia“执行以下代码，并在第 1 行创建一个名为 iteration_number 的变量。此变量的初始值为 1，并且每次执行循环代码时此变量的值都会加 1。最终，当你得到 5 并执行代码时，停止循环”。

for 循环的基本格式为

```
for <iteration_number> in <start_at>:<end_at>
    [your_code]
end
```

术　　语	解　　释
iteration_number	任何有效的变量名，这个变量被称为循环控制变量
start_at	任何数值——值、变量或表达式。这是循环控制变量的初始值
end_at	任何数值——值、变量或表达式。当循环控制变量的值超过此值时，将停止循环。循环控制变量运行 for 和 end 之间的代码，每次到达 end 时，iteration_number 变量的值都会增加，并检查其值是否小于或等于 end_at 中存储的值。如果是，则进行下一次迭代；否则退出循环，并执行 end 后面的代码

为了更好地理解程序是如何工作的，请在 for 循环语句之前添加以下两行代码：

```
1  print("Enter a number to count up to: ")
2  max_number = parse(Int64, readline())
```

然后将 for 循环改为

```
3  for  iteration_number in 1:max_number
4      println(iteration_number)
5  end
```

现在，运行代码，你可以输入一个数字，并看到程序计数到这个数。

例如，如果输入 8，则会看到这样的输出：

```
Enter a number to count up to: 8
1
2
3
4
5
6
7
8
```

就这样，你已经学会了 for 循环的基本知识！但是你还可以做得更多。例如，如果你希望每次都将 iteration_number 增加 2 而不是增加 1，需要做的改变就是

```
for iteration_number in 1:2:max_number
```

现在，如果你在提示符处输入 12，则会看到这样的输出：

```
Enter a number to count up to: 12
1
3
5
7
9
11
```

如果你要打印偶数，则必须从 2 开始。因此，请更改第 1 行为

```
for iteration_number in 2:2:max_number
```

在这里，如果你在提示时输入 12，则会看到这样的输出：

```
Enter a number to count up to: 12
2
4
6
```

```
8
10
12
```

现在，你已经了解了 for 循环的基本知识，让我们开始构建一些更复杂的程序，并将条件与它结合起来。

创建一个名为 stop_at_divisible.jl 的新文件。在此文件中，我们将编写一些能够接收以下输入的代码：

① 起始数；

② 结束数；

③ 整除参考数。

本质上，我们将创建一个 for 循环，从用户提供的第一个数字（起始数）迭代到用户提供的第二个数字（结束数），此循环的增量为 1。每次迭代时，我们都将打印出当前的数。如果数字可以被用户提供的第三个数整除（整除参考数），则停止循环并打印要结束循环的内容。

为了给你一点挑战，我将写出所有代码，然后分块描述它们的作用。

```
1  print("Enter a number to start at: ")
2  start_index = parse(Int64, readline())
3  print("Enter a number to end at: ")
4  end_index = parse(Int64, readline())
5  print("Enter a divisibility reference number: ")
6  div_ref = parse(Int64, readline())
7
8  for iteration_number in start_index:end_index
9      println(iteration_number)
10     if iteration_number % div_ref == 0
11         println("Found the right number! Ending loop.")
12         break
13     end
14 end
```

我相信你可以理解大部分代码，所以让我们只关注第 8 行及之后的

代码。第 8 行定义了 for 循环，它非常简单，它告知 Julia 使用 iteration_number 变量在 start_index 和 end_index 的值之间进行循环迭代，并运行其块内代码。此块在第 14 行(end 关键字)结束。

在该块中，我们首先在第 9 行打印 iteration_number，然后 if 语句检查 iteration_number 是否可被整除参考值整除，这一步使用模(modulo)运算符，这是用百分号(%)表示的。

模运算符返回给定值的除法运算的余数。例如，如果运行

```
15 / 7
```

则会得到

```
2.1428571429
```

但是如果你想得到这个除法的余数而不是一个小数值，则可以使用模运算符，其结果为 1。

如果要手动执行长除，你将得到以下结果：

```
15 / 7
Quotient = 2
Remainder = 1
```

要想证明这一点，你可以做以下计算：

```
7 * 2 + 1 = 15
(15 - 1) / 2 = 7
```

现在请记住这条规则：如果 A 可以被 B 整除，那么 A 除以 B 将返回一个等于 0 的余数。我们可以利用这个规则，如果 A 模 B 是 0，那么 A 就可以被 B 整除。这正是第 9 行的 if 语句检查到的条件。

如果遇到一个在循环中可以整除的数，则将发生以下两件事：

① 该程序将打印出“Found the right number! Ending loop.”

② 该程序将停止循环，这是因为 break 关键字会中断循环中任何剩余的迭代。

现在你已经明白了程序是如何工作的，那就继续吧！你已经赢得了一个当之无愧的休息机会。

我希望你喜欢构建的这个应用程序，但还有更多的内容！让我们通过实现 FizzBuzz 示例继续复习刚刚学习的 for 循环、if 语句和操作符的相关内容吧。

FizzBuzz 是一个经典的编程练习示例，你可以按照以下指示操作：

① 从数字 1 循环到 100；

② 对于每个可以被 3 整除的数字打印“Fizz”；

③ 对于每个可以被 5 整除的数字打印“Buzz”；

④ 对于每个可以同时被 3 和 5 整除的数字打印“FizzBuzz”；

⑤ 对于每个不能被 3 或 5 整除的数字打印当前的数字。

一开始听起来很复杂，但它真的可以只用 11 行 Julia 代码实现。继续操作并打开 FizzBuzz.jl，输入以下代码：

```
1   for iteration_num in 1:100
2       if (iteration_num % 3 == 0) && (iteration_num % 5 == 0)
3           println("FizzBuzz")
4       elseif (iteration_num % 3 == 0)
5           println("Fizz")
6       elseif (iteration_num % 5 == 0)
7           println("Buzz")
8       else
9           println(iteration_num)
10      end
11  end
```

我相信你能理解以上这段代码，它实际上就是刚才五个步骤的实现。

有一个从 1 到 100 的循环，还有一个称为 iteration_num 的迭代器。

循环中的第一个条件检查 iteration_num 是否可以被 3 和 5 同时整除；如果是，则打印 FizzBuzz，如果否，则检查它是否可以被 3 整除；如果是，则打印 Fizz，如果否，则检查它是否可以被 5 整除，如果是，则打印 Buzz，如果否，则打印 iteration_num。

练习

请问以下代码可以实现 FizzBuzz 吗？

```
for iteration_num in 1:100
    if iteration_num % 3 == 0
        print("Fizz")
    end
    if iteration_num % 5 == 0
        print("Buzz")
    end
    if iteration_num % 3 != 0 && iteration_num % 5 != 0
        print(iteration_num)
    end
    print("\n")
end
```

提示：print("\n")可以在屏幕上打印新的一行，换句话说，它会将光标移到下一行。

3.6　如何使用 while 循环迭代

正如你所看到的，for 循环很棒，但有时它们并不能如你所愿。有时，你需要更多的灵活性，这就是 while 循环的作用。使用 while 循环可以在特定条件保持 true 时连续遍历某些代码。下面让我们构建一个简单的示例，从前面讨论的 for 循环中复制一些功能。

创建一个名为 Next_or_quit.jl 的新文件。

```
1  show_next = "c"
2  next_multiple = 0
3
4  while show_next == "c"
5      global next_multiple = next_multiple + 1
6      println(5 * next_multiple)
7      print("Type c to continue, any other letter to stop: ")
8      global show_next = readline()
9  end
10 println("bye")
```

下面对上述代码进行说明，它从创建一个值为 c 的变量 show_next 开始，这里选择字母 c 的意思是 continue，它还创建了另一个值为 0 的变量 next_multiple。第 4 行显示了使用 while 循环的一种方式，从关键字 while 开始，然后是条件

```
show_next == "c"
```

因为我们已经创建了初始值为 c 的变量 show_next，所以条件判断为 true，因此会执行 while 循环中的代码块，即 while 和 end 关键字之间的代码，即从第 5 行到第 8 行。

① 第 5 行会将 next_multiple 的值加 1。

② 第 6 行打印 next_multiple 中的值乘以 5 后的结果。

③ 第 7 行打印一条消息，用户输入 c 可继续，输入任何其他字母则停止。

④ 第 8 行获取用户的输入。请注意，在这个例子中，因为我们需要来自用户的字符串，而不是 Int64，所以没有使用以前用过的 parse(Int64, readline())代码。如果用户想要继续，则需要输入一个字母 c；如果用户想要停止，则需要输任何其他字母。所以我们只需要用到 readline()，而不需要将它转换为 Int64。

⑤ 第 9 行表示 while 循环的结束。此时，控制返回到 while 循环开

始的地方，然后再次判断条件。如果用户输入 c，则条件 show_next == "c"的结果为 true，再次运行代码块。但是，如果用户输 c 之外的任何其他字母，那么代码块就不再被运行。

⑥ 第 10 行将为用户打印消息 bye，如果没有其他迭代，则应用程序将停止。

注意：这与 for 循环不同，它没有像 iteration_number 这样的循环控制变量，因此 while 循环不会运行预定的次数。用户可以继续显示下一个乘 5 运算任意多次，这使得 while 循环既特别又灵活，你也可以使用它代替 for 循环。

但在继续学习之前，我们需要理解一些看起来较新的代码。变量名 show_next 和 next_multiple 之前的 global 告诉 Julia 它引用的是在 while 循环之外，即前面声明过的变量。如果没有使用关键字 global，那么应用程序将会崩溃，因为 Julia 不明白你引用的是 while 循环中的变量还是在该循环之外创建的变量。

每个块都创建了自己的变量，并且只在该块中使用，一旦退出该代码块，这些变量就不复存在了。不过，我们需要在 while 循环的第 1 行和第 2 行使用之前在 while 循环之外声明过的变量及其值，所以需要使用关键字 global。

下面让我们通过示例探索更多使用 while 循环的方法。创建一个名为 simple_while.jl 的新文件，并输入以下代码：

```
1  iteration_num = 0
2
3  while iteration_num < 10
4      global iteration_num += 1
5      println(iteration_num)
6  end
```

在继续学习之前，需要再次强调两点：

① 为什么我们在 iteration_num 之前使用 global 这个词呢？我们将在下一章讨论其真正的复杂性，但简而言之，这是因为我们试图设置在全局范围（即 while 循环外）内定义的变量的值，正如第一个 while 循环示例 Next_or_Quit.jl 中描述的那样。

② 新的操作符“+=”是什么意思呢？使用“+=”操作符可以告诉 Julia 将此运算符左侧的变量值设置为其值加上运算符右侧的值之后的值。

在继续之前，请先查看以下代码（无须输入）：

```
a = 10
a = a + 5
print(a)
```

该代码将打印

```
a = 10
a += 5
print(a)
```

还将打印

```
15
```

因为“+=”告诉 Julia 将变量 a 的值设置为其值加 5。同样，你也可以使用诸如“-=”“*=”“/=”等操作符的速记代码。

现在让我们回到 while 循环。请注意，在 while 关键字的旁边，我们提供了一个条件，while 循环将一遍又一遍地执行代码，直到该条件返回 false，此时循环将停止。因为我们在增加 iteration_num 的值，所以它最终将达到 10，使条件为 false 并停止循环。

但是，上述程序使用 for 循环会更快，那么什么时候才需要使用 while 循环呢？当你要执行一个循环，但不确定需要循环多少次时应该

使用 while 循环。你已经在应用程序 Next_or_Quit.jl 中看到了这样的例子。

好，一旦我们知道如何将变量组合在一起，我们就可以在第 4 章学习 while 循环的更多用法了！

你已经学习了两个基本的编程概念：条件和迭代。现在你应该已经准备好学习更多的中级功能了，其中有更多的类型，并能真正拓宽知识的应用广度。

强化练习

1. 创建一个应用程序，它可以获取用户输入的 3 个数，并按照从小到大的顺序打印。

2. 创建一个程序，它可以遍历用户提供的任意范围的数，并打印除了能被用户提供的一个参考数整除的那些数以外的所有数字。

3. “==”和“=”操作符有什么区别？

4. 创建一个程序，它可以编写任意数字的倍数表，并以正向和反向的顺序显示从 1 到 12 的倍数。

5. 提示用户给出两个正整数 x 和 y，并告知用户较大数是否可以被较小数整除。