第 5 章 求和案例学函数

5.1 动态月份的求和案例

【例 5-1】 数据源是一张成绩表，每月增加一列。需要的结果是对所有月份的成绩求和，如图 5-1 所示。

姓名	1月	2月
甲	10	10
乙	20	20
丙	30	30
丁	40	40

(a) 动态求和案例数据源

= Table.AddColumn(源, "合计", each List.Sum(List.Skip(Record.ToList(_))))

	姓名	1月	2月	合计
1	甲	10	10	20
2	乙	20	20	40
3	丙	30	30	60
4	丁	40	40	80

(b) 最终结果

图 5-1 动态求和案例数据源及最终结果

最终代码如下：

```
//ch5.1 - 01
let
    源 = Excel.CurrentWorkbook(){[Name = "成绩表"]}[Content],
    累计求和 = Table.AddColumn(源, "合计",
        each List.Sum(
            List.Skip(
                Record.ToList(_))))
in
    累计求和
```

5.2～5.6 节都围绕本节的案例展开。

5.2　Table.AddColumn()

5.2.1　界面操作

接5.1节案例，合计列是增加一列，通过界面操作完成。在PQ功能区，单击“添加列”→“自定义列”，弹出的“自定义列”对话框如图5-2所示。

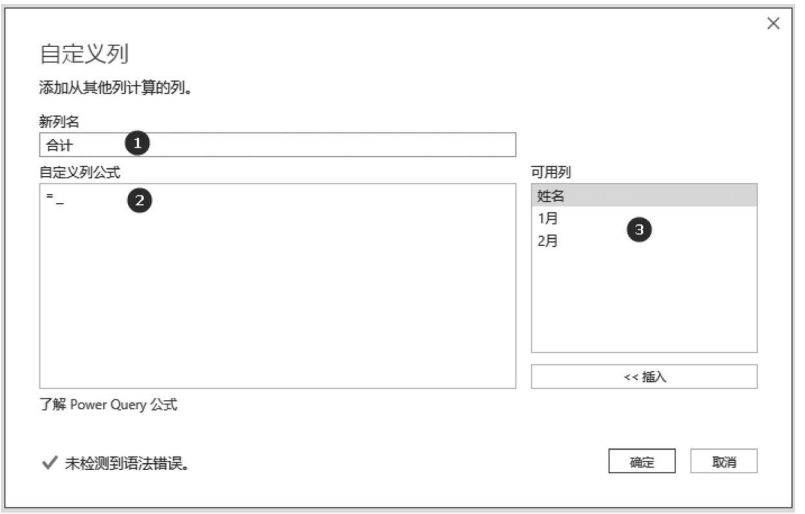


图5-2　“自定义列”对话框

将“新列名”修改为“合计”，在“自定义列公式”框中输入的代码如下：

```
= _
```

PQ会帮助用户自动补全代码，代码如下：

```
= Table.AddColumn(源, "合计", each _ )
```

结果如图5-3所示。

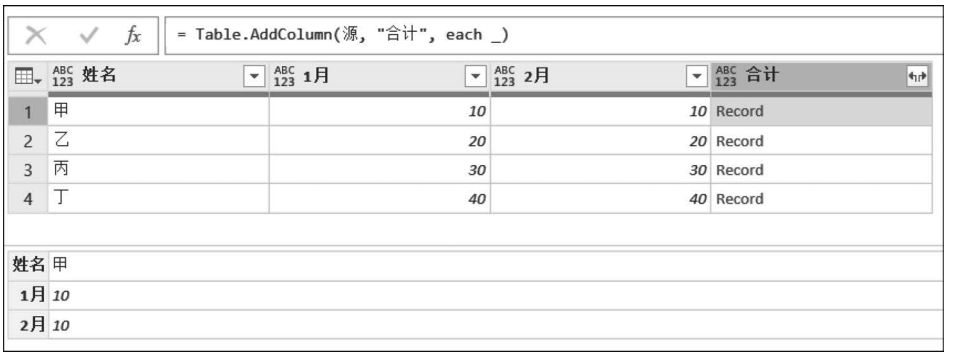


图5-3　添加列的结果

为什么在新增加的一列中每行的值是 record 形式？这是由 Table.AddColumn()的函数传参原理决定的。

5.2.2 理解 record

Table.AddColumn()的作用是在原表上新添加一列，参数如下：

```
Table.AddColumn(
table as table,
newColumnName as text,
columnGenerator as function,
optional columnType as nullable type)
as table
```

等同的代码如下：

```
= Table.AddColumn(源, "合计", each _ )
= Table.AddColumn(源, "合计", (x) => x )
```

第 1 个参数是要新增列的表。

第 2 个参数是新增列的标题，例如“合计”。

第 3 个参数的类型是 function，和 List.Transform()一样，为自定义函数，例如“(x)=>x”或者“each _”。

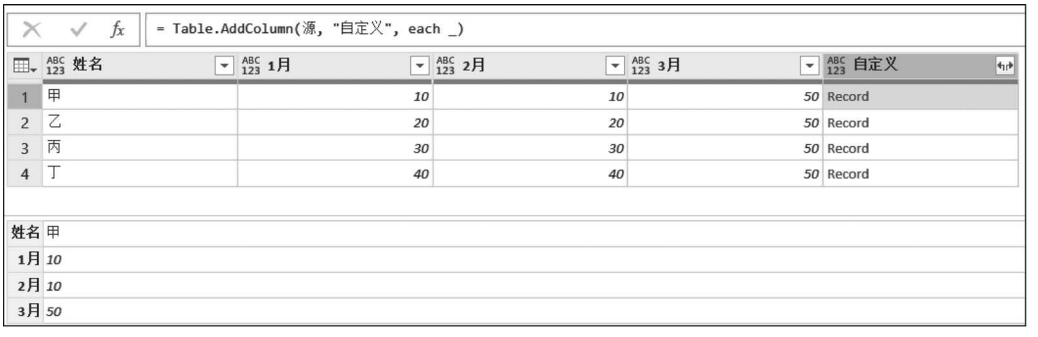
第 4 个参数是可选参数，用于指定新增列的数据类型。

Table.AddColumn()的第 1 个参数是 table，将 table 的每行转换为 record，record 是带标题的行，然后将 record 传递给第 3 个参数。

在第 3 个参数“(x)=>x”中左边的 x 是每行遍历后的 record 形式，再传递到右边的表达式“x”，所以呈现的结果是 record。

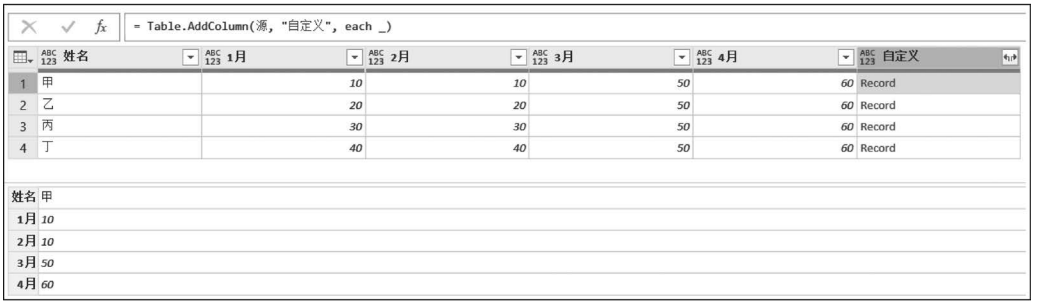
同理，“each _”中的“_”相当于 “(x)=>x”右边的 x，每行遍历后以当前行的 record 形式传递到右边的表达式“_”，每行的结果是 record。

每行的 record 是当前行的映射，如图 5-4 所示。



(a) record映射当前行4列的数据

图 5-4 每行的遍历



(b) record映射当前行5列的数据

图 5-4　（续）

可见，record 是随着数据源列数的增减而动态变化的。

5.2.3　理解第 3 个参数

在对 List. Transform()的讲解中，曾写过代码(x)=>1 和 each 1，同样的写法可以用在 Table. AddColumn()中，因为自定义函数的原理相同，等同的代码如下：

```
= Table.AddColumn(源, "合计", each 1 )
= Table.AddColumn(源, "合计", (x) => 1 )
```

每行的 record 形式仍然传递到第 3 个参数 function 右边的表达式，只是表达式的值是常量 1，所以新列的每行的结果是 1，如图 5-5 所示。

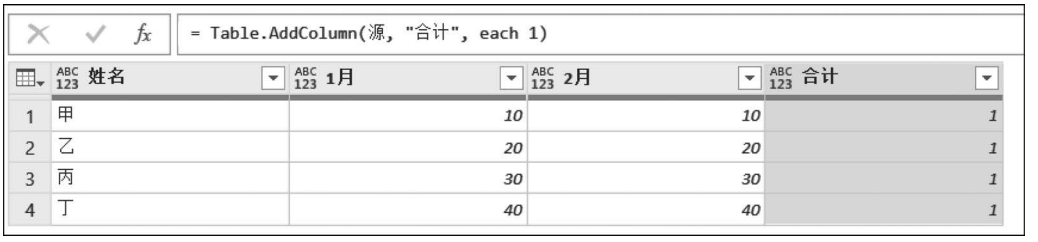


图 5-5　理解第 3 个参数

Table. AddColumn()的第 3 个参数的数据类型是自定义函数，和 List. Transform()的第 2 个参数一样，可以写各种表达式。

注意：Table. AddColumn()的函数名中的 Column 是单数，顾名思义，一次只能添加一列。

5.2.4　each _的简写

因为每行的结果是当前行的 record 形式，所以 record 可进一步深化，等同的代码如下：

```
= Table.AddColumn(源, "合计", each _[姓名])
= Table.AddColumn(源, "合计", (x) => x[姓名])
```

在上述代码中“_”或者 x 代表每行的 record 形式，用“[标题]”可以深化出相应字段的值。

M 函数的灵活应用就是在遍历循环、函数传参的基础上一步步地扩展，例如把姓名列的值取出来后，再连接其他文本，示例代码如下：

```
= Table.AddColumn(源, "合计", each _[姓名] & "同学")
= Table.AddColumn(源, "合计", (x) => x[姓名] & "同学")
```

更加灵活的是“_”可以省略，等同的代码如下：

```
= Table.AddColumn(源, "合计", each _[姓名])
= Table.AddColumn(源, "合计", each [姓名])
= Table.AddColumn(源, "合计", (x) => x[姓名])
```

注意：尽管_和 x 都代表 record，但是只有 each _中的“_”在一定条件下才可以省略，(x)=>x 中右边的 x 不能省略。有“_”必有 each，但有 each 不一定有“_”。

示例代码如下：

```
= Table.AddColumn(源, "合计", each [姓名])
= Table.AddColumn(源, "合计", each 1 )
```

在上述代码中，“each [姓名]”和“each 1”不一样，虽然都没有“_”，但是前者是“each _[姓名]”的省略写法，后者是常量，因为 each 1 是表达式没有用到“_”，所以不写“_”。“[姓名]”之所以能把值取出来，是因为原始形式为“_[姓名]”，即在“_”代表 record 的基础上，把“姓名”这列当前行的值深化出来，理解这点非常重要。

再从界面操作的角度来理解 each _的简写。

在“自定义列”对话框中，在“可用列”栏双击“姓名”列，PQ 会帮助用户自动补全代码，代码如下：

```
= Table.AddColumn(源, "合计", each [姓名] )
```

可见，PQ 自动补全的代码是“each _”的简写形式，如果不理解原理，则无法灵活地扩展应用。例如把 1 月和 2 月的成绩相加，等同的代码如下：

```
= Table.AddColumn(源, "合计", each _[1月] + _[2月] )
= Table.AddColumn(源, "合计", each [1月] + [2月] )
= Table.AddColumn(源, "合计", (x) => x[1月] + x[2月] )
```

前文讲过，null 与任何数值进行加、减、乘、除运算的结果都为 null，因此，上述代码改成 List.Sum()更好，以避免空值造成结果错误，等同的代码如下：

```
= Table.AddColumn(源, "合计", each List.Sum({_[数学],_[语文]} )
= Table.AddColumn(源, "合计", each List.Sum({ [数学], [语文]} )
= Table.AddColumn(源, "合计", (x) => List.Sum({x[数学],x[语文]} )
```

“_”能够省略的条件是当“each _”中的“_”代表 record 或 table 并用标题进行深化时(同样地,只有源头是 record 和 table 时,才能用标题进行深化),可以省略“_”,等同的代码如下:

```
//ch5.2 - 01
= List.Transform({[a = 1,b = 2],[a = 2,b = 2]},each _[a])
= List.Transform({[a = 1,b = 2],[a = 2,b = 2]},each [a])
```

否则结果完全不同,示例代码如下:

```
= List.Transform({{1,2},{3,4}},each _{0})          //深化索引
= List.Transform({{1,2},{3,4}},each {0})           //创建 list
```

结果如图 5-6 所示。

(a) 深化索引　　(b) 创建list

图 5-6　省略“_”的结果

能够省略“_”的原因是深化标题的形式为“_[标题]”,创建 record 的形式是“[标题=值]”,省略“_”在函数传递过程中不会产生歧义。

5.3　Record.ToList()

自本节开始,用简化的传参形式写代码,不再同时写 3 种传参形式(each _、(x)=>x、each)。

接 5.2 节的案例结果,当前行的 record 形式已经获取,代码如下:

```
= Table.AddColumn(源, "合计", each _)
```

现在有 1 月、2 月的成绩,进行累计求和,代码如下:

```
= Table.AddColumn(源, "合计", each List.Sum({[1 月],[2 月]}))
```

上述代码可得到当前已有月份的求和,但无法动态求和。这种代码称为“硬代码”。当原始数据增加了将来月份的列时不可能每个月都修改上述代码。

书写灵活的代码，仍从 record 入手。“each _”中的“_”代表 record，思路是把 record 所有字段的值取出，再用 List.Sum()求和。用到的函数是 Record.ToList()。

Record.ToList()的作用是取出 record 所有字段对应的值，返回 list，参数如下：

```
Record.ToList(record as record) as list
```

Record.ToList()只有一个参数，类型是 record，示例代码如下：

```
= Record.ToList([a = 1, b = 2])     //{1,2}
```

“_”代表的是 record，可作为 Record.ToList()的参数，代码如下：

```
= Record.ToList(_)
```

“_”是当前行的 record 形式，随着月份的增加，“_”包含的月份数是动态增加的，因此，上述代码可动态地列出值。

Record.ToList()将 record 转换成 list，是否有其他函数用于各个容器的转换？答案是有的，同类函数有 Record.FromList()、Record.ToTable()、Table.ToColumns()等十几个函数。这是一大类函数，其作用像魔方一样，将表、行、列扭转后重新归位，参见第 13 章。

5.4 Record.FieldValues()

M 函数非常多，5.3 节中用 Record.ToList()将 record 中所有字段的值取出，本节介绍 Record.FieldValues()，也能获得同样的结果。

Field 是字段、Value 是值。

Record.FieldValues()的作用是取出 record 中所有字段对应的值，返回 list，参数如下：

```
Record.FieldValues(record as record) as list
```

Record.FieldValues()只有一个参数，类型是 record，示例代码如下：

```
= Record.FieldValues([a = 1, b = 2])     //{1,2}
```

5.5 List.Skip()

在 5.3 节案例中，通过 Record.ToList()将 record 所有字段对应的值取出，构成了一个 list。例如第 1 行的值是“{"甲",10,10}”，如果这个 list 直接作为 List.Sum()的第 1 个参数，则会报错，因为“甲”是文本，不能参与数值的求和计算，接下来，用 List.Skip()删除 list 中索引 0 对应的元素。

List.Skip()的作用是将 list 中最前面的符合条件的元素删除后返回列表，参数如下：

```
List.Skip(
list as list,
optional countOrCondition as any)
as list
```

第 1 个参数的类型是 list。

第 2 个参数是可选参数，类型是数字或 function。

1. 默认第 2 个参数

第 2 个参数的默认值为 1，可以省略。等同的代码如下：

```
= List.Skip({1,2,3})              //{2,3}
= List.Skip({1,2,3},1)            //{2,3}
= List.Skip({1,2,3},null)         //{2,3}
```

当第 2 个参数是 null 时，表示没有第 2 个参数，即相当于第 2 个参数是 1 的结果。

2. 第 2 个参数是数字

第 2 个参数可以是数字，例如 1、2 等，数字代表删除前 n 个值，示例代码如下：

```
= List.Skip({1,2,3},2)          //{3}
= List.Skip({1,2,3},8)          //{} 删除的元素个数超过总元素个数，则返回空列表
```

3. 特别用法

第 1 个参数可以是空列表，第 2 个参数为任何值都返回空列表，代码如下：

```
= List.Skip({})
= List.Skip({},8)
= List.Skip({},null)
```

4. 第 2 个参数是 function

第 2 个参数也可以是自定义函数，参数要求里的 Condition 指条件判断，即表达式的值为布尔值。示例代码如下：

```
= List.Skip({1,2,3,4,3,3,4},each _< 3)
= List.Skip({1,2,3,4,3,3,4},(x) => x < 3)
```

结果如图 5-7 所示。

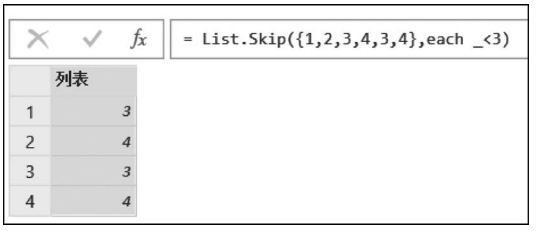


图 5-7　List.Skip()的第 2 个参数的用法

List.Skip()先对第 1 个参数 list 中的每个元素进行循环遍历，然后传递到第 2 个参数，再根据＝＞右边的表达式进行逻辑判断，如果逻辑值是 true，则 skip(删除、跳过)，当遇到 false 时，跳出循环。循环遍历的逻辑见表 5-1。

表 5-1 List.Skip()循环遍历的逻辑

第 1 个参数的元素	第 2 个参数的表达式(x)=>x<3(或 each _<3)	结 果
1	(1)=> 1<3	true
2	(2)=> 2<3	true
3	(3)=> 3<3	false
4		
3		
4		

当遍历到第 3 个元素时,如果结果是 false,则跳出循环,删除前两个元素,从第 3 个元素开始保留,从而构成一个 list。

示例代码如下:

```
= Table.Skip({1..10},each true)      //{}
```

在上述代码中,因为每次循环遍历的结果都是 true,所以删除所有元素,结果返回空列表。

List.Skip()和 List.Select()的相同点是第 2 个参数 function 的表达式的值为布尔值。不同点是 List.Select()用于遍历所有元素,Select(筛选)出表达式值为 true 对应的元素,从而构成 list; List.Skip()从第 1 个元素开始遍历,如果遍历到结果为 false 对应的元素,则跳出循环,Skip(删除)最前面条件判断结果为 true 对应的元素,保留后面的元素,从而构成 list,后面的元素不再遍历。

根据 List.Skip()的用法,修改 5.3 节案例,代码如下:

```
= Table.AddColumn(源, "合计",
    each List.Sum(
        List.Skip(
            Record.ToList(_))))
```

5.6 案例总结

在 Table.AddColumn()中,"each _"中的"_"代表当前行的 record 形式,当当前行是 2 月份时,record 是[姓名="甲", 1 月=10, 2 月=10],当当前行是 3 月时,record 是[姓名="甲", 1 月=10, 2 月=10,3 月=10],代码灵活地实现了在数据源增加月份后都能取到当前行所有的数据。通过 Record.ToList()取出 record 所有字段对应的值,通过 List.Skip()删除姓名列的值,最后用 List.Sum()求和。

这个案例是动态横向求和的通用思路。

5.7 List.First()

List.Skip()的同类函数,见表 5-2。

表 5-2　List.Skip()的同类函数

函　数　名	参　　数
List.First	(list as list, optional defaultValue as any) as any
List.Last	(list as list, optional defaultValue as any) as any
List.FirstN	(list as list, countOrCondition as any) as list
List.LastN	(list as list, countOrCondition as any) as list
List.RemoveFirstN	(list as list, optional countOrCondition as any) as list
List.RemoveLastN	(list as list, optional countOrCondition as any) as list

List.First()的作用是返回 list 的第 1 个元素，参数如下：

```
List.First(
list as list,
optional defaultValue as any)
as any
```

第 1 个参数的类型是 list。

第 2 个参数是可选参数，是当第 1 个参数为空列表时的返回值。示例代码如下：

```
= List.First({0..5})                    //0
= List.First({0..5}, -1)                //0
= List.First({null,null}, -1)           //null
= List.First({})                        //null
= List.First({}, -2)                    //-2
```

List.First()和 List.Last()的参数和使用方法相同。List.Last()的作用是返回 list 的最后一个元素。

5.8　List.FirstN()

List.FirstN()、List.LastN()、List.RemoveFirstN()、List.RemoveLastN()的遍历、传参逻辑与 List.Skip()相通。当条件判断的结果为 false 时，跳出循环。List.Skip()和 List.RemoveFirstN()的效果相同。示例代码如下：

```
//ch5.8-01
= List.FirstN({},2)                          //{ }
= List.FirstN({1..5},2)                      //{1,2}
= List.FirstN({1..5},each _<3)               //{1,2}
= List.LastN({1..5},each _>3)                //{4,5}
= List.RemoveFirstN({1..5},each _<3)         //{3,4,5}
= List.RemoveLastN({1..5},each _>2)          //{1,2}
```

以 List.RemoveLastN()为例，循环遍历的逻辑见表 5-3。

表 5-3 List.RemoveLastN()循环遍历的逻辑

第 1 个参数的元素	第 2 个参数的表达式 each _>2	结　果
1		
2	each 2>2	false
3	each 3>2	true
4	each 4>2	true
5	each 5>2	true

从 list 的最后一个元素开始遍历并进行逻辑判断，当结果为 true 时，相对应的元素被 Remove(删除)，直到结果为 false 时，跳出循环，保留后面对应的元素，从而构成一个 list。

5.9 求累计金额案例

【例 5-2】 计算截至当月的累计销售额，如图 5-8 所示。

图 5-8 累计求和案例数据源

这个需求在 Excel 中容易实现，用混合引用锁定求和的范围。PQ 中的混合引用通过深化、索引实现，相对于 Excel 函数略复杂。

5.10 Table.AddIndexColumn()

接 5.9 节案例，PQ 中实现混合引用，先添加索引列，再取出前 n 行。

Table.AddIndexColumn()的作用是在原表上添加索引列，参数如下：

```
Table.AddIndexColumn(
table as table,
newColumnName as text,
optional initialValue as nullable number,
optional increment as nullable number,
optional columnType as nullable type)
as table
```

添加索引列的界面操作如图 1-1 所示，代码如下：

```
= Table.AddIndexColumn(源, "索引", 1, 1)
```

第 1 个参数是要添加索引列的表。

第 2 个参数是索引列的标题。

第 3 个参数是可选参数，是初始的第 1 个索引值，省略时默认为 0。

第 4 个参数是可选参数，是索引值增量，当省略时默认为 1。

第 5 个参数是可选参数，用于表示索引列的值的数据类型，并且在 Excel 2016 版本中没有第 5 个参数。

当只写必选参数时相当于第 3 个参数为 0，第 4 个参数为 1，等同的代码如下：

```
= Table.AddIndexColumn(源, "索引")
= Table.AddIndexColumn(源, "索引", 0, 1)
```

索引值增量指相邻的两个索引值的差，例如，要创建的索引列的值为{1,3,5,7,…}，代码如下：

```
= Table.AddIndexColumn(源, "索引", 1, 2)
```

将如图 5-8 所示的数据添加索引列后，再新增 1 个自定义列，把第 1 个步骤“源”的表引用过来，代码如下：

```
let
    源 = Excel.CurrentWorkbook(){[Name = "金额表"]}[Content],
    索引 = Table.AddIndexColumn(源, "索引", 1, 1),
    结果 = Table.AddColumn(索引, "累计", each 源)
in
    结果
```

结果如图 5-9 所示。

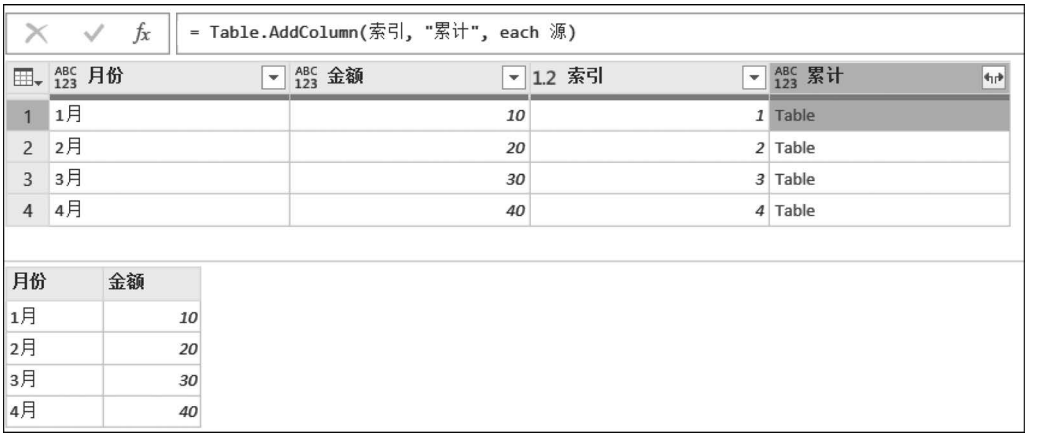


图 5-9　添加源表

这时，每行的值 Table 都是相同的表。下一步将深化金额列，代码如下：

```
结果 = Table.AddColumn(索引, "累计", each 源[金额])
```

结果如图 5-10 所示。

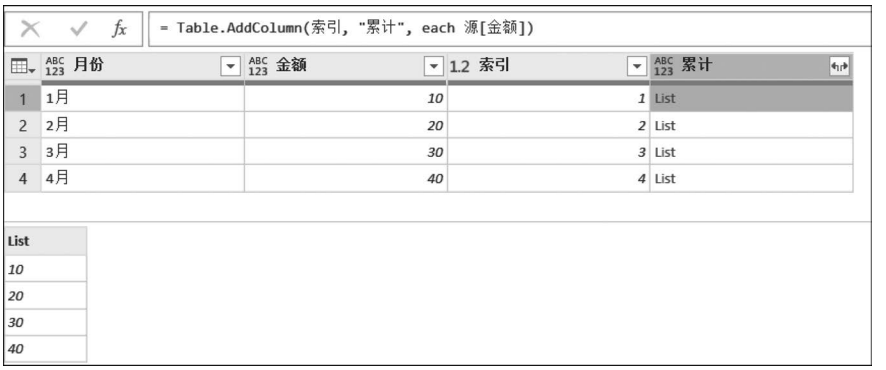


图 5-10　深化源表

思考一下，下面两行代码的区别：

```
结果 = Table.AddColumn(索引, "累计", each 源[金额])
结果 2 = Table.AddColumn(索引, "累计", each [金额])
```

在第 1 行代码中，“源[金额]”的作用相当于 table[标题]，用于深化出第 1 个步骤的表的金额列，结果是 list，每行的“源[金额]”是固定值。

在第 2 行代码中，“[金额]”是“_[金额]”的简写，是从当前行的 record 形式深化出来的，所以这个“[金额]”是当前行金额列的值，每行的值不同。

下一步，根据索引列获取 list 的前 n 个值，代码如下：

```
结果 = Table.AddColumn(索引, "累计", each
        List.FirstN(源[金额],[索引]))
```

在上述代码中“[索引]”是“_[索引]”的简写，是从当前行的 record 形式深化出来的，每行的值不同，因此，每行 List.FirstN()的第 2 个参数依次是 1、2、3 等，从而获得了混合引用值区域的效果，结果如图 5-11 所示。

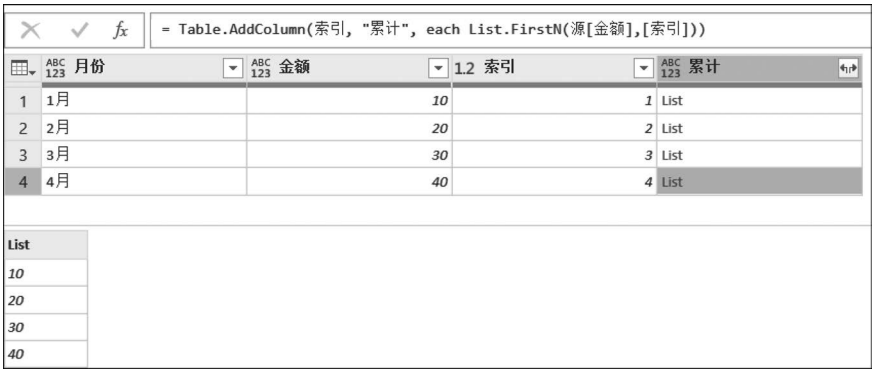


图 5-11　用索引获取 list 的元素

最后，对每行的 list 求和，最终代码如下：

```
//ch5.10 - 01
let
    源 = Excel.CurrentWorkbook(){[Name = "金额表"]}[Content],
    索引 = Table.AddIndexColumn(源, "索引",1,1),
    结果 = Table.AddColumn(索引,"累计", each
            List.Sum(List.FirstN(源[金额],[索引])))
in
    结果
```

结果如图 5-12 所示。

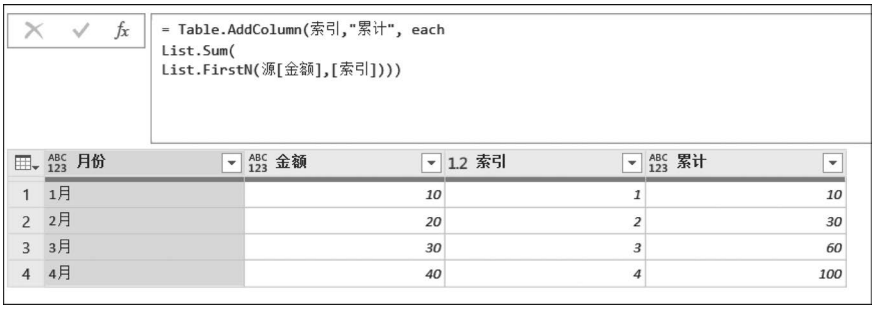


图 5-12　累计求和的结果

在 PQ 中，添加索引列是求累计值的通用思路。通过本节案例复习了 Table.AddColumn()和其他函数的嵌套、三大容器深化和 each 传参的原理。

5.11　PQ 技巧

5.11.1　界面操作

本书用的界面操作以 PQ 功能区为主，在实操中可以根据个人的使用习惯，也可以通过在 PQ 编辑区右击后在弹出的菜单中操作，如图 5-13 所示。

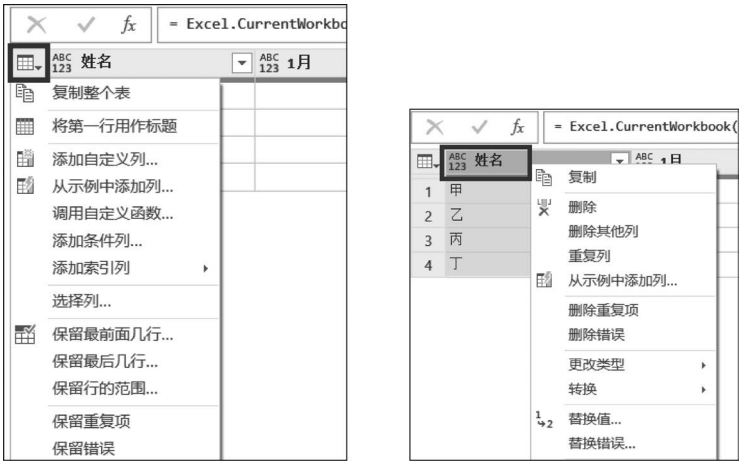


图 5-13　右击后弹出的菜单

5.11.2 快速获取数据

数据源如图 5-14 所示。

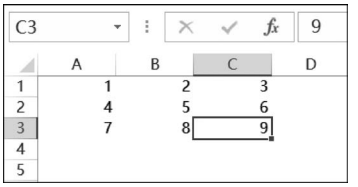


图 5-14 数据源

A1～C3 的范围，不论是普通区域还是超级表区域，在任意一个单元格右击，在弹出的菜单中单击“从表格/区域获取数据”，这个范围将作为超级表导入 PQ 中，其作用相当于在 Excel 功能区单击“数据”→“来自表格/区域”，如图 5-15 所示。

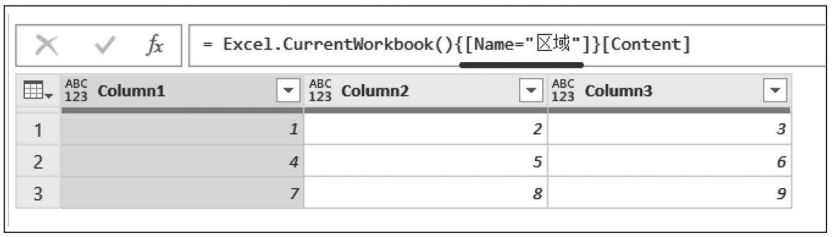
图 5-15 快捷地将数据导入 PQ 中

“来自表格/区域”中的“表格”指超级表，超级表的翻译来自 Table。“区域”指代什么范围？选中 A1～C3 单元格，操作步骤如图 5-16 所示。

(a) 选中区域，定位名称框

图 5-16 定义名称区域

(b) 修改名称，以两种方法导入PQ



(c) 导入PQ的结果

图 5-16　（续）

可见，定义名称区域和超级表一样都是有名称的区域，可以导入 PQ 中。