第 3 章

M 函数基础

基于前两章的基础知识，本章学习一些简单的常用的 M 函数。

PQ 中有多少个 M 函数？笔者的 Excel 365 版本中有 830 个。和 Excel 函数一样，PQ 的函数也不用全部学完。学习方法是了解有多少类函数，以及每类函数中常用的函数有哪些。了解函数之间的通性，当遇到某个需求时搜索相关的 M 函数。

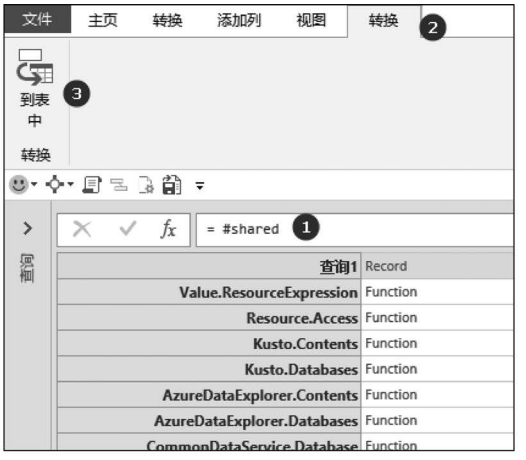
本书通过类比法，将有通性的函数结合起来讲解。

3.1 查询所有 M 函数

在 PQ 中，查询所有 M 函数的代码如下：

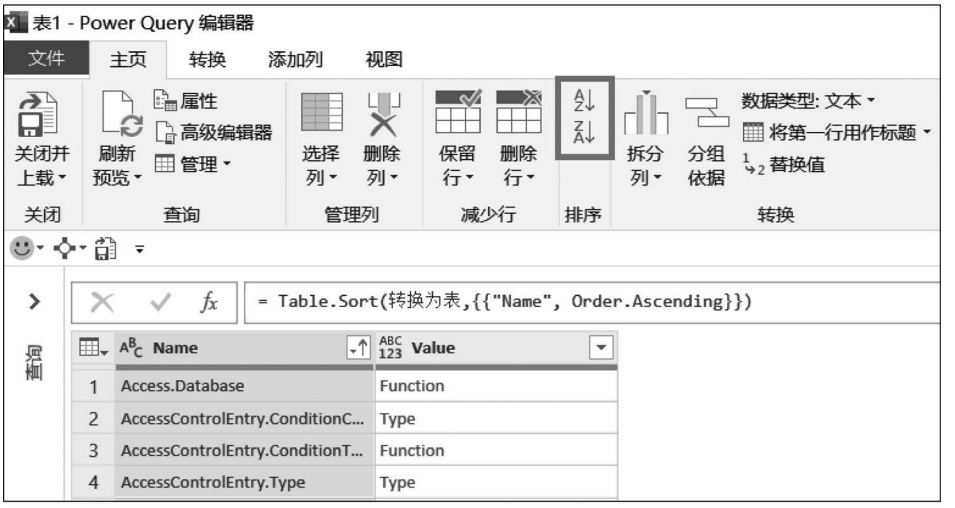
```
= #shared
```

上述代码返回的结果是以 record 形式列出所有的 M 函数及当前所有的查询名（例如查询 1），单击“转换”→“到表中”，将 record 转换成 table，对 Name 列排序，结果如图 3-1 所示。



(a) 所有的M函数

图 3-1　查询所有的 M 函数



(b) 对Name列排序

图 3-1　（续）

在所有类名中，Table 类、Record 类、List 类、Text 类、Date 类、Number 类、文件类的相关函数使用频率高。

只导出当前 PQ 编辑器中所有的查询名，代码如下：

```
= #sections
```

3.2　聚合函数

在 Excel 中，函数按照类别进行了分类，如图 3-2 所示。

图 3-2　Excel 函数

在 PQ 中，函数是按照类名分类的，多个聚合的函数在 List 类中。笔者用的 Excel 365 版本中有 72 个 List 类函数。本节学习简单的常用的聚合函数。

3.2.1　List.Sum()

List.Sum()的作用是返回 list 中非空值的总和。如果列表中没有非空值，则返回 null。参数如下：

```
List.Sum(
list as list,
optional precision as nullable number)
as any
```

示例代码如下：

```
= List.Sum({1,2,3,null})              //6
= List.Sum({null,null})               //null
= List.Sum({})                        //null
```

第 1 个参数的类型是 list。

第 2 个参数是可选参数，用于精度控制。

当列表中没有元素时，是一个空列表，代码如下：

```
= { }
```

3.2.2 List.Average()

List.Average()的作用是忽略空值，返回 list 中各项的平均值。如果列表为空，则返回 null，参数如下：

```
List.Average(
list as list,
optional precision as nullable number)
as any
```

示例代码如下：

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

第 1 个参数的类型是 list。

第 2 个参数是可选参数，用于精度控制。

3.2.3 List.Min()

List.Min()的作用是忽略空值，返回 list 中各项的最小值。如果列表为空，则返回可选默认值，可以指定一个返回值。参数如下：

```
List.Min(
list as list,
optional default as any,
optional comparisonCriteria as any,
optional includeNulls as nullable logical)
as any
```

示例代码如下：

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

第 1 个参数的类型是 list。

第 2 个参数是可选参数，表示在第 1 个参数的 list 为空或 list 元素都是 null 时，返回的默认值。

第 3 个参数和第 4 个参数是可选参数。

同类函数有 List.Max()，参数和使用方法与 List.Min()相同。

3.2.4　List.MinN()

List.MinN()的作用是忽略空值，返回 list 中最小的 N 项。参数如下：

```
List.MinN(
list as list,
countOrCondition as any,
optional comparisonCriteria as any,
optional includeNulls as nullable logical)
as list
```

示例代码如下：

```
= List.MinN({1,2,3,4,5},2)                    //{1,2}
= List.MinN({1,2,3,4,5,null,null},2)          //{1,2}
```

第 1 个参数的类型是 list。

第 2 个参数是数字或比较条件，当为数字时，用于指定返回最小的 N 项。

第 3 个参数和第 4 个参数是可选参数。

同类函数有 List.MaxN()，参数和使用方法与 List.MinN()相同。

3.2.5　List.Count()

List.Count()的作用是返回 list 中元素的个数，参数如下：

```
List.Count(list as list) as number
```

示例代码如下：

```
= List.Count({1,2,3})              //3
= List.Count({1,2,3,null})         //4
= List.Count({})                   //0
```

只有一个参数，类型是 list。

3.2.6　List.NonNullCount()

List.NonNullCount()的作用是返回 list 中非空元素的个数，参数如下：

```
List.NonNullCount(list as list) as number
```

示例代码如下：

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

只有一个参数，类型是 list。

3.2.7 List.Product()

List.Product()的作用是返回 list 中非空元素的乘积。如果列表中没有非空值，则返回 null，参数如下：

```
List.Product(
list as list,
optional precision as nullable number)
as nullable number
```

示例代码如下：

```
= List.Product({1,2,3})                   //6
= List.Product({1,2,3,null})              //6
= List.Product({null,null})               //null
= List.Product({})                        //null
```

第 1 个参数的类型是 list。

第 2 个参数是可选参数，用于精度控制。

观察本节的 M 函数，第 1 个参数的数据类型都是 list。大多数 List 类函数的第 1 个参数是 list。用上述类比法，当遇到其他计算需求时，可在所有 M 函数中查找单词。例如求众数，找到函数 List.Mode()或 List.Modes()；求标准偏差，找到函数 List.StandardDeviation()；求中位数，找到函数 List.Median()。

3.3 空值 null

在 3.2 节的 M 函数中，特别提到对空值的处理。当空值从 Excel 中导入 PQ 中时将显示为 null。

在 Excel 的单元格中，肉眼不可见的值有多种，只有真空，导入 PQ 后才显示为 null，如图 3-3 所示。

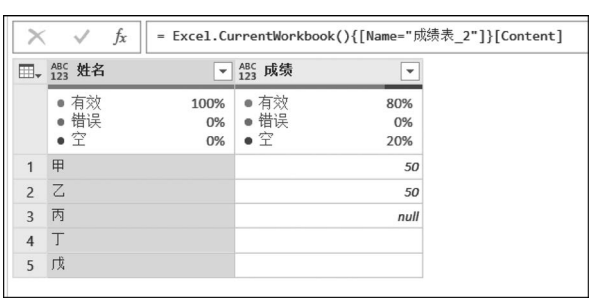


图 3-3 真空与不可见字符

例如空文本(假空),当导入 PQ 中时不为 null。Excel 和 PQ 中的空文本,代码如下:

```
= ""
```

一个值中有一个或多个空格,或者其他不可见打印字符,在 PQ 中不显示为 null。这些细节在实操中很常见,只要不显示为 null,而又看不见内容,就说明值为空文本占位或者其他不可见字符,可用 Text. Trim()、Text. Clean()、Text. Remove()等方法清洗。

3.4　关键字

null 是 PQ 的关键字之一。关键字是内置的类似标识符的字符序列,已被赋予特别的含义,在未转义的情况下,不能被用于步骤名。null 作为步骤名的错误提示如图 3-4 所示。

图 3-4　null 是关键字不能用作步骤名

PQ 中的关键字见表 3-1。

表 3-1　PQ 中的关键字

关　键　字	关　键　字	关　键　字	关　键　字
and	or	not	if
else	then	true	false
let	in	try	otherwise
each	null	type	as
is	error	meta	section
shared	#table	#binary	#date
#time	#datetime	#datetimezone	#duration
#infinity	#nan	#shared	#sections

可见,在 3.1 节用到的#shared 和#sections 是关键字。

3.5　运算符和标点符号

PQ 中的运算符和标点符号如下:

```
, ; = <<= >>= <> + - * / & ( ) [ ] { } @ ! ? ?? => .. ...
```

除了圆括号、方括号、花括号、逗号,其他常用符号的作用如下。

(1) 算术运算符:+(加)、-(减)、*(乘)、/(除)。

(2) 比较运算符:=(相等)、<(小于)、>(大于)、>=(大于或等于)、<=(小于或等

于)、<>(不等于)。

(3) 赋值运算符:=。

(4) & 连接符,可连接文本、table、list、record、null 等,示例代码如下:

```
= {1,2} & {3,4}                 //{1,2,3,4}
= [a = 1, b = 2] & [c = 3]      //[a = 1,b = 2,c = 3]
= [a = 1, b = 2] & [b = 3]      //[a = 1,b = 3] 相同字段的值被后面的 record 替代
= null & null                   //null
= "1" & "2"                     //"12"
```

(5) =>用于函数的参数传递,非常重要,参见第 4 章。

(6) ?和??是深化的语法糖,用于简化语法,参见 18.12 节。

(7) @用于递归。

(8) …是 error 的快捷方式,参见 18.13 节。

(9) ..是创建 list 连续元素的快捷方式,使用频繁,本节重点讲解。

3.5.1 连续的列表

建立连续的列表,可用..快捷方法,代码如下:

```
= {开始字符..结束字符}
```

(1) 创建连续的数字列表,代码如下:

```
= {1..5}              //{1,2,3,4,5}
= {100..0}            //{} 只能是正序的列表
= {1..1}              //{1}
```

数字列表的最小值是-2 147 483 648,最大值是 2 147 483 647,list 中最多容纳 2 147 483 647 个数。数值上下限和元素个数应同时满足,示例代码如下:

```
= {-2147483649.. -2147483648}
//Expression.Error: 数量超出 32 位整数值范围.详细信息:-2147483649
= {2147483647.. 2147483648}
//Expression.Error: 数量超出 32 位整数值范围.详细信息:2147483648
=List.Count({-2147483648.. -2})
//2147483647
={-2147483648.. -1}
//{} 超过元素个数限制,返回空列表
```

(2) 创建连续的数字型文本字符,只能创建单字符列表,代码如下:

```
= {"0".."9"}            //正确写法
= {"0".."11"}           //11 是双字符,错误提示如图 3-5 所示
```

要实现从文本 1 到文本 11 的创建,用 List.Transform()循环遍历。

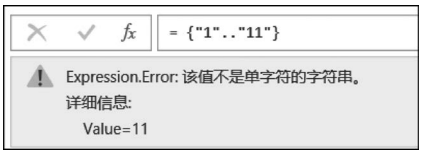


图 3-5　双字符的错误提示

(3) 创建常见的汉字列表,代码如下:

```
={"一".."龟"}
```

(4) 创建英文字母列表,代码如下:

```
={"a".."z"}                    //创建小写字母列表
={"A".."Z"}                    //创建大写字母列表
={"A".."Z","a".."z"}           //创建大小写字母列表,用逗号分开列表元素
```

创建大小写字母为何不写{"a".."Z"}或{"A".."z"}。这种快速创建方法基于 Unicode 编码的原理,参见 20.1 节。

3.5.2　加减乘除不简单

在日常清洗中,加、减、乘、除的使用非常频繁。只有细节满分,才能保证运算结果正确。进行加、减、乘、除运算最好使用函数,而不是运算符,因为运算符不支持空值的处理。

(1) 数字与 null 的运算。

在 Excel 中,空值当作 0 处理。数 1 是数字,数 2 是空值,运算结果如图 3-6 所示。

数1	数2	数1+数2	数1*数2	数1/数2	数2/数1
1		1	0	#DIV/0!	0

图 3-6　Excel 中空值的计算

在 PQ 中,任何数值和空值用运算符进行加、减、乘、除运算,结果是 null,代码如下:

```
= 1 + null          //null
= 1 - null          //null
= null - 1          //null
= 1 * null          //null
= 1 / null          //null
= null / 0          //null
```

最佳实践,代码如下:

```
= List.Sum({1, null})              //1
= List.Sum({ - 成绩,null})
//成绩在此处代表一个变量,减法也用 List.Sum()实现
= List.Product({1, null})          //1
```

(2) 数字与 0 相除。

在 Excel 中,0 和空值的处理一样。在 PQ 中,0 作为除数,结果是 infinity,表示无穷大,

代码如下：

```
= 1 / 0          //infinity、∞、正无穷大
= -1 / 0         //-infinity、-∞、负无穷大
```

(3) 0 与 0 运算。

在 Excel 中，0 和空值的处理一样。在 PQ 中，0 和 0 相除，结果是 NaN，即 Not a Number，代码如下：

```
= 0 / 0      //NaN
```

当结果是 NaN 或 infinity 时，容错处理的代码如下：

```
//ch3.5-01
let
  源 = 数1/数2,                    //数1、数2代表某个数字或者空值
  容错 =
    if Number.IsNaN(源)
      or 源 = 1/0                  //1/0表示正无穷大
      or 源 = -1/0                 //-1/0表示负无穷大
    then null
    else 源
in
  容错
```

上述代码实现的效果是，当数 2 是 0，或者数 1、数 2 都是 0 时，结果不再是 infinity 或 NaN，而是 null。

总之，null 与任何值用运算符进行加、减、乘、除运算，其结果都是 null，最好用 List.Sum()等函数替代运算符。

Number.IsNaN()的作用是判断数字是否为 NaN，代码如下：

```
= Number.IsNaN(0/0)      //true
```

3.6 if 条件语句

在 ch3.5-01 代码中，用到的 if then else 是条件语句。熟悉条件语句，可以从界面操作开始。在 PQ 功能区，单击“添加列”→“条件列”，弹出的“添加条件列”对话框如图 3-7 所示。

条件语句和 Excel 中的 IF()作用相同。Excel 中的 IF()用逗号分隔参数，PQ 中的 if 语句用关键字 if、else、then 和空格分隔语句，其有两种写法。

1. 双分支

双分支命令如下：

```
if true then 结果1 else 结果2
```

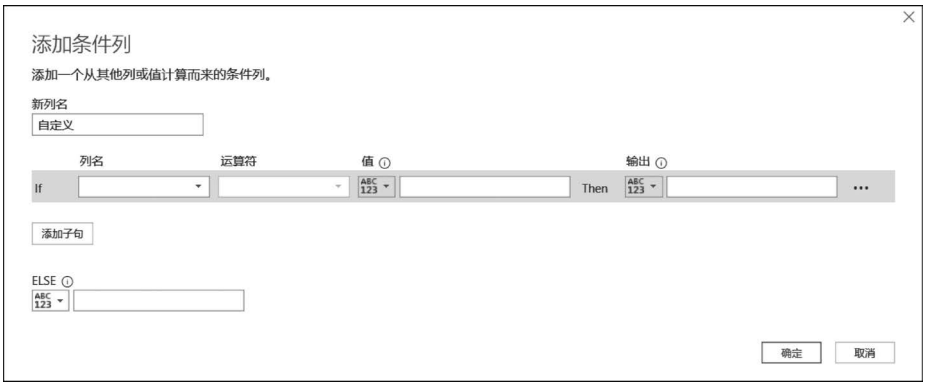


图 3-7　“添加条件列”对话框

示例代码如下：

```
let
    成绩 = 90,
    评比 = if 成绩>=90 then "优秀" else "合格"
in
    评比
```

上述语句的作用是，如果成绩>=90，则返回“优秀”，否则返回“合格”。双分支语句是从两个结果中返回其中的一个结果。

2. 多分支

多分支命令如下：

```
if true then 结果 1
else if true then 结果 2
else 结果 3
```

示例代码如下：

```
//ch3.6-01
let
    成绩 = 90,
    评比 = if 成绩>=90 then "优秀"
        else if 成绩>=80 then "良好"
        else if 成绩>=60 then "合格"
        else "不及格"
in
    评比
```

上述语句的作用是，如果成绩>=90，则返回“优秀”；如果成绩>=80，则返回“良好”；如果成绩>=60，则返回“合格”，否则返回“不及格”。多分支是从多个结果中返回其中的一个结果。

编程语言中都有 if 语句，初学 PQ 容易和其他语言的 if 语句的语法混淆。if 语句的关键字都是小写，并且 else if 是分开书写的。Python 中的写法是 elif。

if 后面的表达式的值必须是布尔值 true 或 false。

3.7 布尔值

布尔值是 true 和 false，也是关键字，是逻辑表达式的值。

条件语句 if 后面是布尔值，布尔值可以通过逻辑表达式得出。如果表达式的值不是布尔值，则会造成语法错误，示例错误代码如下：

```
= if 1 then "结果是 1" else "结果是 2"        //错误提示,无法将 1 转换为逻辑类型
```

修改代码如下：

```
= if 1 = 1 then "结果是 1" else "结果是 2"
```

1＝1 是逻辑判断，结果是 true。同理，修改代码如下，也是成立的。

```
= if true then "结果是 1" else "结果是 2"
= if false then "结果是 1" else "结果是 2"
```

true 与 1(非 0)对应，false 与 0 对应。PQ 中相关的逻辑函数，示例代码如下：

```
= Logical.From(1)              //true
= Logical.From( - 100)         //true
= Logical.From(0)              //false
= Logical.FromText("true")     //true
= Logical.FromText("True")     //true
= Logical.ToText(true)         //"true"
= Number.From(true)            //1
```

3.8 逻辑运算符

逻辑运算符有 or、and、not，它们都是关键字，示例代码如下：

```
= 条件 1 and 条件 2 and 条件 n
= 条件 1 or 条件 2 or 条件 n
= not true                  //false
= not false                 //true
```

or 和 and 用于连接多个条件判断，每个条件的表达式的值是布尔值或 null，示例代码如下：

```
//ch3.8 - 01
let
    数学 = 98,
    语文 = 98,
    条件语句 =
        if 数学> = 90 and 语文> = 90 then "全科优秀"
        else if 数学> = 90 or 语文> = 90 then "单科优秀"
        else "无优秀科目"
in
    条件语句
```

and 的运算逻辑是当所有条件均成立时，结果为 true；如果有一个条件不成立，则结果为 false。可以理解为，true 是 1，false 是 0，and 是 *（乘法）。例如，true and false and false，结果为 1 * 0 * 0，因此，只要有一个 false，其结果为 false。

or 的运算逻辑是当任一个条件成立时，结果为 true，如果所有条件均不成立，则结果为 false。可以理解为，true 是 1，false 是 0，or 是＋（加法）。例如，true or false or false，结果为 1＋0＋0，因此，只要有一个 true，结果为 true。

根据上述逻辑，and 只要遇到 false，最终结果就为 false；or 只要遇到 true，最终结果就为 true。得出确定结果后，逻辑运算符后面的表达式无须继续运算。这种计算方式称为短路运算。

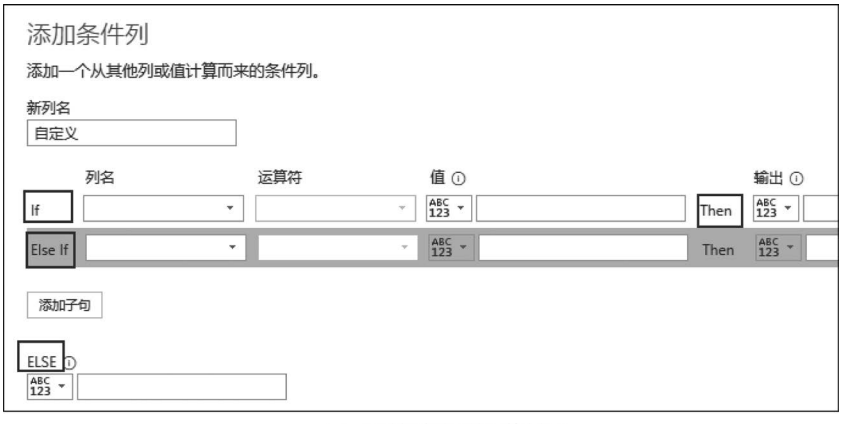
not 的示例代码如下：

```
//ch3.8 - 02
let
    成绩 = 98,
    条件语句 =
        if not (成绩>=90) then "未达优秀" else "优秀"
in
    条件语句
```

注意：在一个分支语句中，当 not、and 和 or 混合使用时，建议加上括号，避免逻辑混乱，从而造成结果不符合预期，参见 23.6 节。

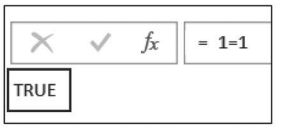
3.9　显示的误区

在 PQ 中数据类型、数据结构、关键字都是小写单词，但是在界面上显示出来时，有时是大写，有时是小写，并不代表在 M 函数中可以任意书写，示例如图 3-8 所示。

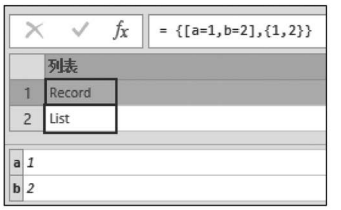


(a) “添加条件列”的显示

图 3-8　大小写的显示问题



(b) 逻辑值的显示



(c) 数据结构的显示

图 3-8 （续）

PQ 中这种显示方式容易造成误解，读者要注意区别，参见 3.10 节。

3.10 is 和 = 的用法

本节讲解关键字 is 的用法，实操中经常用到，代码如下：

```
//注意数据类型、数据结构、关键字都是小写
= {1,2,3} is list
= "A" is text
= 123 is number
= [a = 1, b = 2] is record
= #date(2023,1,1) is date
= #datetime(2023,1,1,12,0,0) is datetime
```

is 用于判断表达式的值是否为某种数据类型/结构，结果返回布尔值。

“=”不能用于判断数据类型/结构，错误代码如下：

```
= {1,2,3} = list
= "A" = text
= 123 = number
```

“=”用于比较两个表达式的值是否相等，代码如下：

```
= {1,2,3} = {1,3,2}                 //false
= [a = 1, b = 2] = [b = 2, a = 1]   //true
= 123 = "123"                       //false
```

通过上述代码可以看出，由 table 和 record 标题的唯一性决定了标题的顺序改变，table 和 record 的值不变；由于 list 有索引，所以当 list 中的元素顺序改变时 list 的值改变。

注意：a={1,2,3}={2,3,4}，第 1 个“=”是赋值运算符，将右边的表达式赋值给变量 a，第 2 个“=”是比较运算符，两个 list 的比较结果为 false。

3.11 try 容错语句

当表达式结果出现错误时，将提示 Error，导致后面的运算或步骤无法进行，如图 3-9 所示。

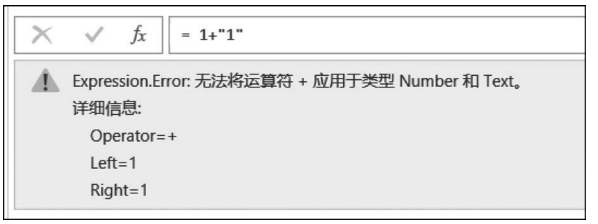


图 3-9　表达式出现错误时的提示

很多情况下，运算结果 Error 是正常的过程，如果想要的结果是跳过该 Error，继续进行下一次运算，则可使用容错语句，格式和示例代码如下：

```
= try 表达式
= try 1 + "1"
```

try 的结果是 record，如图 3-10 所示。

图 3-10　try 的结果是 record

判断表达式是否有错误的代码如下：

```
= (try 1 + "1")[HasError]      //true
```

处理容错，使用频率更高的是 try otherwise。格式和示例代码如下：

```
= try 表达式 1 otherwise 表达式 2
= try 1 + "1" otherwise "先转换数据类型再运算"
```

如果表达式 1 有错误，则返回表达式 2，否则返回表达式 1。等同的运算逻辑代码如下：

```
= try 表达式 1 otherwise 表达式 2
= if (try 表达式 1)[HasError] then 表达式 2 else 表达式 1
```

实操中，容错语句使用频繁，try 后面的表达式 1 和 otherwise 后面的表达式 2 可在各种场景中灵活地应用。

3.12　#的用法

#的作用有多种。

1. 声明关键字

例如#table,用于区别#table()和 table,代码如下:

```
//ch3.12-01
= #table({"姓名","成绩"},{{"甲",10},{"乙",20}}) is table     //true
```

2. 对关键字转义

例如 null 是关键字,当 null 用于步骤名时会报错,转义后可使用。此处只用于说明用法,关键字作为步骤名并非最佳实践。示例代码如下:

```
#"null" = 1
```

3. 输出特殊字符

例如回车符#(cr)、换行符#(lf)、制表符#(tab)。示例代码如下:

```
//ch3.12-02
= "abc" & "#(lf)" & "dfg"
```

结果如图 3-11 所示。

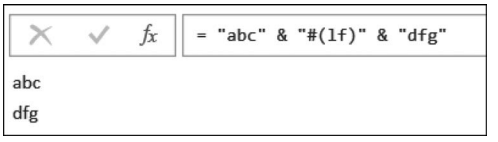


图 3-11 特殊字符

当有多个特殊字符需要输出时,示例代码如下:

```
//ch3.12-03
= "abc" & "#(lf,cr)" & "dfg"
= "abc" & "#(lf)#(cr)" & "dfg"
```

4. 转义 4 位和 8 位十六进制

#(000D)、#(0000000D)和#(cr)是等效的。

3.13 总结

M 函数的知识量看似繁多,其实环环相扣。通过学习原理、内在逻辑来理解这些知识点,层层递进,只要多写、多练习,就能掌握其用法。

单独讲解某个函数或者某类函数比较枯燥,当实战中遇到需求时,可能想不到用哪个函数。第 4 章讲解 M 函数的参数传递原理,这样便可以通过案例来讲解函数的灵活应用了。