第 3 章

M 语言基础

3.1 Excel 函数

在 Power Query 中，M 语言函数一般会包含两部分，第一部分通常表示函数所在的类别，第二部分通常表示函数的功能。例如，List.Transform()函数，类别为 List，功能函数为 Transform()。参见第 1 章的表 1-1，Excel 中的 Power Query M 语言函数共分为 125 个类别。其实，Excel 的函数也是有类别的，只不过 Excel 函数的类别是隐性的。

3.1.1 Office 支持

Excel 中所有函数的类别可以通过微软官方网站或 Excel 内部功能按钮查看。

1. Excel 函数(按类别列出)

以下是如何查看 Excel 的函数类别。打开浏览器，在搜索中输入“Excel 函数 按类别”，单击“百度一下”按钮，便可显示微软官方相应网页，如图 3-1 所示。

图 3-1 Excel 函数(按类别)查询

在微软“Excel 函数(按类别列出)”网页，Excel 函数被划分为 14 类，如图 3-2 所示。

2. Excel 函数(选择类别)

Excel 函数的类别也可以直接从 Excel 中获知。打开 Excel，单击编辑栏左侧的“插入函数”(fx)图标，在弹出的“插入函数”对话框中可以找到 Excel 函数的所有类别，如图 3-3 所示。

图 3-2　Excel 函数(按类别列出)

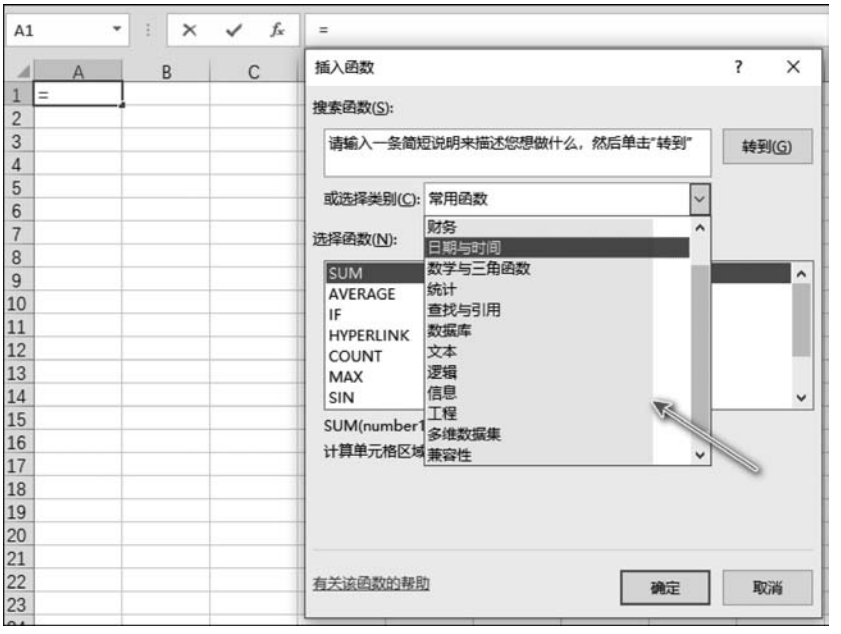

图 3-3　Excel 函数的类别

在图 3-3 中可供选择的 Excel 函数的类别有“财务、日期与时间、数学与三角函数、统计、查找与引用、数据库、文本、逻辑、信息、工程、多维数据集、兼容性、与加载项一起安装的用户定义的函数、Web 函数”，其划分的标准与图 3-2 是一致的。

3.1.2　Excel 函数汇总

查找 Excel 函数的类别，可采用“按类别”方式，也可采用“按字母”方式。打开浏览器，在搜索中输入“Excel 函数 按字母”，单击“百度一下”按钮，便可显示微软官方相应网页，如图 3-4 所示。

图 3-4　Excel 函数(按字母)查询

1. Excel 函数(按字母顺序)

在微软官网“Excel 函数(按字母顺序)”页面，复制官网页面网址，完成代码如下：

```
//ch301 - 001
let
    herf = "https://support.microsoft.com/zh - cn/office/excel - 函数 - 按字母顺序 - b3944572 - 255d - 4efb - bb96 - c6d90033e188",

    源 = Web.Page(Web.Contents(herf)){0}[Data],

    分列 A = Table.SplitColumn(
                源,
                "函数名称",
                Splitter.SplitTextByEachDelimiter(
                        {" "},
                        QuoteStyle.Csv, false
                ),
                {"函数名称"}
            ),

    分列 B = Table.SplitColumn(
                分列 A,
```

```
                "类型和说明",
                Splitter.SplitTextByAnyDelimiter
                  (
                    {"#(00A0)",": "," "},
                    QuoteStyle.Csv
                  ),
                {"函数(类别)"}
    ),

    替换 = Table.TransformColumns(分列 B,
              {
                "函数(类别)", each
                 List.Accumulate(
                {
                    {"函数",""},
                    {"和","与"}
                },
                _ ,
                (x,y) => Text.Replace(x,y{0},y{1})
                )
              }
    ),

    分组 = Table.Group(
                替换,
                "函数(类别)",
                {
                    {"Excel 函数", each Text.Combine(
                        List.Distinct(
                        List.Sort([函数名称])),
                       " , ")
                },
                    {"个数", each Table.RowCount(_)}
                }
    ),

    排序 = Table.Sort(分组,{"个数",1})

in
    排序
```

通过以上代码，可以获取网页中所有 Excel 函数及其类别划分，如表 3-1 所示。

表 3-1 Excel 的所有函数

函数(类别)	Excel 函数	个数
统计	AVEDEV, AVERAGE, AVERAGEA, AVERAGEIF, AVERAGEIFS, BETA. DIST, BETA. INV, BINOM. DIST, BINOM. DIST. RANGE, BINOM. INV, CHISQ. DIST, CHISQ. DIST. RT, CHISQ. INV, CHISQ. INV. RT, CHISQ. TEST, CONFIDENCE. NORM, CONFIDENCE. T, CORREL, COUNT, COUNTA, COUNTBLANK, COUNTIF, COUNTIFS, COVARIANCE. P, COVARIANCE. S, DEVSQ, EXPON. DIST, F. DIST, F. DIST. RT, F. INV, F. INV. RT, F. TEST, FISHER, FISHERINV, FORECAST, FORECAST. ETS, FORECAST. ETS. CONFINT, FORECAST. ETS. SEASONALITY, FORECAST. ETS. STAT, FORECAST. LINEAR, FREQUENCY, GAMMA, GAMMA. DIST, GAMMA. INV, GAMMALN, GAMMALN. PRECISE, GAUSS, GEOMEAN, GROWTH, HARMEAN, HYPGEOM. DIST, INTERCEPT, KURT, LARGE, LINEST, LOGEST, LOGNORM. DIST, LOGNORM. INV, MAX, MAXA, MAXIFS, MEDIAN, MIN, MINA, MINIFS, MODE. MULT, MODE. SNGL, NEGBINOM. DIST, NORM. DIST, NORM. S. DIST, NORM. S. INV, NORMINV, PEARSON, PERCENTILE. EXC, PERCENTILE. INC, PERCENTRANK. EXC, PERCENTRANK. INC, PERMUT, PERMUTATIONA , PHI, POISSON. DIST, PROB, QUARTILE. EXC, QUARTILE. INC, RANK. AVG, RANK. EQ, RSQ, SKEW, SKEW. P, SLOPE, SMALL, STANDARDIZE, STDEV. P, STDEV. S, STDEVA, STDEVPA, STEYX, T. DIST, T. DIST. 2T, T. DIST. RT, T. INV, T. INV. 2T, T. TEST, TREND, TRIMMEAN, VAR. P, VAR. S, VARA, VARPA, WEIBULL. DIST, Z. TEST	111
数学与三角	ABS, ACOS, ACOSH, ACOT, ACOTH, AGGREGATE, ARABIC, ASIN, ASINH, ATAN, ATAN2, ATANH, BASE, CEILING. MATH, CEILING. PRECISE, COMBIN, COMBINA, COS, COSH, COT, COTH, CSC, CSCH, DECIMAL, DEGREES, EVEN, EXP, FACT, FACTDOUBLE, FLOOR. MATH, FLOOR. PRECISE, GCD, INT, ISO. CEILING, LCM, LET, LN, LOG, LOG10, MDETERM, MINVERSE, MMULT, MOD, MROUND, MULTINOMIAL, MUNIT, ODD, PI, POWER, PRODUCT, QUOTIENT, RADIANS, RAND, RANDARRAY, RANDBETWEEN, ROMAN, ROUND, ROUNDDOWN, ROUNDUP, SEC, SECH, SEQUENCE, SERIESSUM, SIGN, SIN, SINH, SQRT, SQRTPI, SUBTOTAL, SUM, SUMIF, SUMIFS, SUMPRODUCT, SUMSQ, SUMX2MY2, SUMX2PY2, SUMXMY2, TAN, TANH, TRUNC	80

续表

函数(类别)	Excel 函数	个数
财务	ACCRINT, ACCRINTM, AMORDEGRC, AMORLINC, COUPDAYBS, COUPDAYS, COUPDAYSNC, COUPNCD, COUPNUM, COUPPCD, CUMIPMT, CUMPRINC, DB, DDB, DISC, DOLLARDE, DOLLARFR, DURATION, EFFECT, FV, FVSCHEDULE, INTRATE, IPMT, IRR, ISPMT, MDURATION, MIRR, NOMINAL, NPER, NPV, ODDFPRICE, ODDFYIELD, ODDLPRICE, ODDLYIELD, PDURATION, PMT, PPMT, PRICE, PRICEDISC, PRICEMAT, PV, RATE, RECEIVED, RRI, SLN, SYD, TBILLEQ, TBILLPRICE, TBILLYIELD, VDB, XIRR, XNPV, YIELD, YIELDDISC, YIELDMAT	56
工程	BESSELI, BESSELJ, BESSELK, BESSELY, BIN2DEC, BIN2HEX, BIN2OCT, BITAND, BITLSHIFT, BITOR, BITRSHIFT, BITXOR, COMPLEX, CONVERT, DEC2BIN, DEC2HEX, DEC2OCT, DELTA, ERF, ERF. PRECISE, ERFC, ERFC. PRECISE, GESTEP, HEX2BIN, HEX2DEC, HEX2OCT, IMABS, IMAGINARY, IMARGUMENT, IMCONJUGATE, IMCOS, IMCOSH, IMCOT, IMCSC, IMCSCH, IMDIV, IMEXP, IMLN, IMLOG10, IMLOG2, IMPOWER, IMPRODUCT, IMREAL, IMSEC, IMSECH, IMSIN, IMSINH, IMSQRT, IMSUB, IMSUM, IMTAN, OCT2BIN, OCT2DEC, OCT2HEX	54
兼容性	BETADIST, BETAINV, BINOMDIST, CEILING, CHIDIST, CHIINV, CHITEST, CONFIDENCE, COVAR, CRITBINOM, EXPONDIST, FDIST, FINV, FLOOR, FTEST, GAMMADIST, GAMMAINV, HYPGEOMDIST, LOGINV, LOGNORMDIST, MODE, NEGBINOMDIST, NORM. INV, NORMDIST, NORMSDIST, NORMSINV, PERCENTILE, PERCENTRANK, POISSON, QUARTILE, RANK, STDEV, STDEVP, TDIST, TINV, TTEST, VAR, VARP, WEIBULL, ZTEST	40
文本	ARRAYTOTEXT, ASC, BAHTTEXT, CHAR, CLEAN, CODE, CONCAT, CONCATENATE, DBCS, DOLLAR, EXACT, FIND, FINDB, FIXED, JIS, LEFT、LEFTB, LEN、LENB, LOWER, MID、MIDB, NUMBERVALUE, PHONETIC, PROPER, REPLACE、REPLACEB, REPT, RIGHT、RIGHTB, SEARCH、SEARCHB, SUBSTITUTE, T, TEXT, TEXTJOIN, TRIM, UNICHAR, UNICODE, UPPER, VALUE, VALUETOTEXT	35
查找与引用	ADDRESS, AREAS, CHOOSE, COLUMN, COLUMNS, FILTER, FORMULATEXT, GETPIVOTDATA, HLOOKUP, HYPERLINK, INDEX, INDIRECT, LOOKUP, MATCH, OFFSET, ROW, ROWS, RTD, SORT, SORTBY, TRANSPOSE, UNIQUE, VLOOKUP, XLOOKUP, XMATCH	25
日期与时间	DATE, DATEDIF, DATEVALUE, DAY, DAYS, DAYS360, EDATE, EOMONTH, HOUR, ISOWEEKNUM, MINUTE, MONTH, NETWORKDAYS, NETWORKDAYS. INTL, NOW, SECOND, TIME, TIMEVALUE, TODAY, WEEKDAY, WEEKNUM, WORKDAY, WORKDAY. INTL, YEAR, YEARFRAC	25
信息	CELL, ERROR. TYPE, INFO, ISBLANK, ISERR, ISERROR, ISEVEN, ISFORMULA, ISLOGICAL, ISNA, ISNONTEXT, ISNUMBER, ISODD, ISREF, ISTEXT, N, NA, SHEET, SHEETS, TYPE	20

续表

函数(类别)	Excel 函数	个数
Database	DAVERAGE, DCOUNT, DCOUNTA, DGET, DMAX, DMIN, DPRODUCT, DSTDEV, DSTDEVP, DSUM, DVAR, DVARP	12
逻辑	AND, FALSE, IF, IFERROR, IFNA, IFS, NOT, OR, SWITCH, TRUE, XOR	11
多维数据集	CUBEKPIMEMBER, CUBEMEMBER, CUBEMEMBERPROPERTY, CUBERANKEDMEMBER, CUBESET, CUBESETCOUNT, CUBEVALUE	7
Web	ENCODEURL, FILTERXML, WEBSERVICE	3
加载项与自动化	CALL, EUROCONVERT, REGISTER. ID	3

统计表 3-1 中的所有 Excel 函数的个数，共计 481 个，其中常用的 10 个函数为 SUM()、IF()、LOOKUP()、VLOOKUP()、MATCH ()、CHOOSE()、DATE ()、DATES ()、FIND()、INDEX()。

回顾 1.2.1 节，在 Excel 中的 Power Query 的 M 语言函数为 808 个。由此可见 M 语言函数的数量远多于 Excel 函数的数量，而且通过后续的深入学习，读者会越来越清晰地发现：除 M 语言中的容器类(Table、List、Record)、数据库类(Sql、Sap、Access 等)、时间智能函数，其余大量的 M 语言函数其实是与 Excel 函数语法与功能极其类似的。M 语言真正学习的重点在于数据类型及数据结构的转换，在这 808 个函数中，常用的函数不到 30 个。

基于以上的发现：读者完全可以采取“求同存异”的原则，“同”是指一些具备共性、共通的知识点。先消化 Excel 与 M 语言共通的函数(例如，文本类函数、数学与三角类函数、日期与时间函数等)，然后掌握 M 语言的一些共性知识(例如，复杂函数名称的关键字构成法、常用参数的数值化含义)，再进阶容器类(Table、List、Record)及数据库类(Sql、Access 等)函数的学习，从而降低学习曲线的陡峭度。

2. Excel 函数的构成

在 Excel 中运用饼图，统计表 3-1 中的数据，如图 3-5 所示。

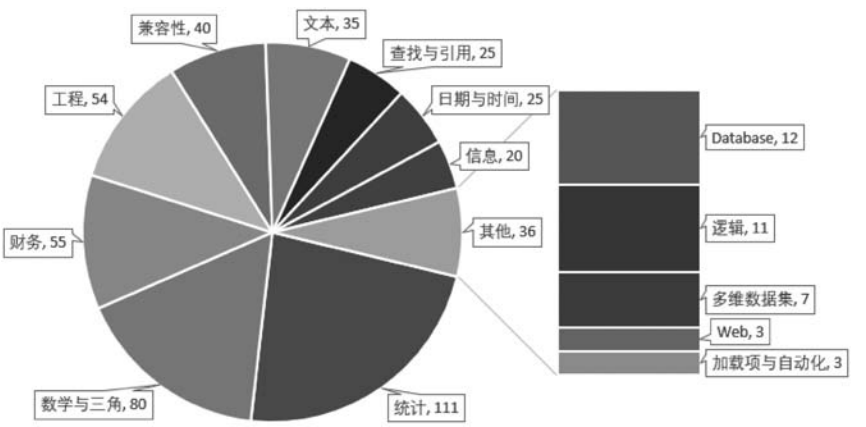

图 3-5 Excel 函数(按类别统计)

3. Excel 函数与 M 语言函数

很多 M 语言函数与 Excel 函数的语法与功能是极其类似的，如表 3-2 所示。

表 3-2 Excel 函数与 M 语言函数的类别对应（举例）

Excel 函数的类别	Excel 函数	M 语言函数	M 语言函数的类别
文本	Len	Text. Length()	Text
数学与三角	Round	Number. Round()	Number
信息	Iseven	Number. IsEven()	Number
逻辑	if	if	关键字
日期与时间	Date	#date()	Date
	Now	DateTime. LocalNow()	DateTime
统计	Count	Record. FieldCount()	Record
		List. Count()	List
		Table. RowCount()	Table
查找与引用	Match	List. MatchesAll()	List
		Table. MatchesAllRows()	Table

从表 3-2 的对应情况来看，Excel 中的“文本”类别的函数多对应于 M 语言中的 Text 类别的函数；“数学与三角、信息”类别的函数多对应于 M 语言中的 Number 类别的函数；“日期与时间”类别的函数多对应于 M 语言中的 Date、DateTime 等类别的函数；“统计、查找与引用”类别的函数多对应于 M 语言中的 List、Record、Table 等类别的函数。

3.2 M 语言函数

3.2.1 M 语言函数简介

1. M 语言函数（按类别列出）

以下是如何查看 Power Query M 语言函数的类别。打开浏览器，在搜索中输入“Power Query M 函数”（大小写不敏感），单击“百度一下”按钮，便可显示微软官方相应网页，如图 3-6 所示。

图 3-6 Power Query M 函数搜索

在微软“Power Query M 函数参考”网页，M 语言函数被划分为 24 类，如图 3-7 所示。

2. M 语言函数的框架性理解

从理论上来讲，很多计算机语言的程序都可理解为程序＝数据＋方法。Power Query 的 M 语言也可以这样理解。

图 3-7　M 语言函数(按类别列出)

在“程序＝数据＋方法”理论中，“数据”有“数据结构”与“数据类型”之分。在 Power Query 中，“数据结构”从狭义范围来讲，主要是指“表、记录、列表”三大容器；从广义范围来讲，它还包括参数中的“合并器、拆分器、比较器、替换器”等。“数据类别”从狭义范围来讲，主要是指“文本、数字、日期时间、逻辑”等；从广义范围来讲，它还涵盖“表、记录、列表”等数据结构，如表 1-1 中所列举的 Table、Record、List、Text、Number、Date、Logical 等类别。

“方法”(method)主要是指“M 语言函数”及“语法规则”，它涉及“逻辑运算、增、删、改、查、分组聚合、数据处理、报错兼容”等方面。以“文本”类别的函数为例，它有“合并、拆分、移除、替换、重复”等功能，对应的函数有 Text.Combine()、Text.Split()、Text.Remove()、Text.Replace()、Text.Repeat()等。在很多函数中，读者可以通过指定“区间范围、位置、分隔符”等方式进行更多复杂场景的应用。为加深读者对 M 语言的知识体系的框架性理解，本书将围绕图 3-8 中的内容进行展开。

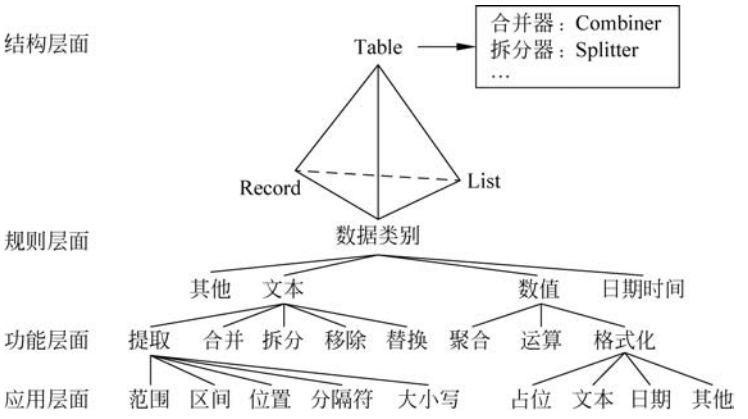

图 3-8　Power Query M 语言的体系

3. M语言函数名称的共性总结

常有使用者认为“M语言类别众多、函数冗长、参数复杂、易忘难学”，简化记忆、降低理解的难度是学习M语言之初首先要考虑的问题。

以下是可以降低M语言学习难度的几种方法：

参照图3-8，从整体上了解M语言的结构体系；参照图3-7，从整体上了解M语言的函数类别；参照表3-2，从整体上了解M语言函数与Excel函数类别间的对应关系。

既然“会写M语言函数的鄙视会改M语言函数的，会改M语言函数的鄙视只会界面操作的”，那么何不先熟悉M语言界面操作后自动生成的代码，然后对其改写；改写的程度由浅入深，在不断熟记与掌握M语言函数语法的基础上，逐步深入理解M语言的核心。

在改写Power Query高级编辑器自动生成的代码的过程中，参照图1-25中的关键字，对所有函数的构成进行共性总结，找出M语言的命名规律。参照图3-8，找出三大容器(Table、Record、List)间的结构转换规律，以及各容器内数据类型的转换(To、From)的规律，不断地对代码量进行压缩、压缩、再压缩，最终实现化繁为简。

参照图1-25，采用“函数 按字母”的方式，对图1-25中的单词进行函数命名的拼接，最终完成知识的转化过程。经对比图1-25后发现，以下单词在常用的M语言函数中使用的频率较高，如表3-3所示。

表3-3 M语言函数中名称的关键字

首字母	关键字
A	Add、Alternate、Any、All
S	Select、Split、Skip、Sort、Start
T	Table、To、Transform、Trim、Transpose、Type、Text、Time
R	Range、Record、Remove、Replace、Repeat、Rows、Reorder、Rename、Reverse
C	Combine、Columns、Contains、Count、Clean
M	Matching、Matches
P	Pivot、Promote、Position、 Proper、Pad
I	Is、Index、Insert、Item

通过表3-3，细心的读者会发现：在M语言中，60%以上的常用函数是由这些单词组成的。例如，Table. AddColumn()、Table. AlternateRows()、Table. SelectRows()、Table. SplitColumn()、Table. Skip()、Table. ToRows()、Table. TransformRows()、Table. TransformColumns()……

通过以上的单词组合后，细心的读者不难发现：所谓冗长的M语言函数名称，其实是由一些最常见、高频的单词依据一定的规则拼接而成的，而且含有某些特定单词的函数其参数中往往有其一定的特征。

例如，函数中含有Is单词的函数有55个，其返回值为true或false，常用于条件判断；函数中含有Contains单词的函数有7个，其返回值为true或false，常用于条件判断。

例如，函数中含Combine单词的函数，大概率有一个参数指定数据为list数据结构。

而对于那些函数名称拼写简单的函数，有可能在添加 M 语言的类别后，其函数的名称与用法同 Excel 中的名称与用法完全一致或类似。例如，Text. Replace()、Text. Repeat()、Text. Trim()。

由此可见，Power Query M 语言虽然函数众多，但其构建规则仍有迹可循，稍做共性总结后立即能找出其中的规律。

4. 可常量化参数(0、1、2)

在 Power Query M 语言中，0、1、2 常用作某些函数参数的别名。在别名化过程中这 3 个参数所代表的意义如下：

0 代表默认值。例如，升降序中的升序、表查询时的内连接、分组时的全局分组、位置查询时的首次出现位置、查询无结果时系统的自动报错功能等。

1 代表常见方式。例如，升降序中的降序、表查询时的左外部连接、分组时的局部分组、位置查询时的末次出现位置、查询无结果时系统的忽略错误功能等。

2 代表使用空值或所有位置显示之类的功能，即全有或全无之类的。

以下是一些常用参数与对应的 0、1、2 别名，如表 3-4 所示。

表 3-4 常量值参数及其代表的意义(1)

适用函数	适用场景	0 代表的意义	1 代表的意义	2 代表的意义
List. Sort, Table. Sort	Sort order	Order. Ascending	Order. Descending	
List. DateTimes, List. Dates, List. DateTimeZones, List. Durations, List. Generate, List. Numbers, List. Random	Occurrence specification	Occurrence. First	Occurrence. Last	Occurrence. All
Record. ToTable, Record. FromTable, Record. ToTable	MissingField	MissingField. Error	MissingField. Ignore	MissingField. UseNull
Table. SplitColumn, Table. FromList	Splitter	ExtraValues. List	ExtraValues. Error	ExtraValues. Ignore
Table. Group	GroupKind	GroupKind. Local	GroupKind. Global	

5. 可常量化参数(0～6)

常量值参数及其代表的意义如表 3-5 所示。

6. 不可常量化的参数

Compare 的 3 种比较方式：Compare. Ordinal 区分大小写，Compare. OrdinalIgnoreCase 不区分大小写，Compare. FromCulture 区域语言选项。

Replacer 有两种替换方式：Replacer. ReplaceText 替换局部字符串(不支持数值)，Replacer. ReplaceValue 替换完整的值(匹配字符串或数值)。

表 3-5 常量值参数及其代表的意义(2)

可常量化参数	表的连接方式	表的连接算法
0	JoinKind. Inner	JoinAlgorithm. Dynamic
1	JoinKind. LeftOuter	JoinAlgorithm. PairwiseHash
2	JoinKind. RightOuter	JoinAlgorithm. SortMerge
3	JoinKind. FullOuter	JoinAlgorithm. LeftHash
4	JoinKind. LeftAnti	JoinAlgorithm. RightHash
5	JoinKind. RightAnti	JoinAlgorithm. LeftIndex
6		JoinAlgorithm. RightIndex

3.2.2 语法差异

在学习新语言的过程中,由于对语法的理解不到位,遇到报错提示是常有的事情。读者完全可以利用这些语法报错的机会借以历练、总结与成长。以下是一些在 Excel 与 M 语言使用过程中常见的语法差异及对应的报错提示。

1. 表达式错误

在 Excel 中,文本与数值可以直接进行文本拼接。在单元格中,表达式=3&"A"的返回值为"3A",而在 Power Query 中,表达式=3&"A"返回的错误提示如下:

```
Expression.Error: 无法将运算符 & 应用于类型 Number 和 Text。
详细信息:
    Operator = &
    Left = 3
    Right = A
```

出错原因:表达式是在给定环境中的运算。在 M 语言中,文本与数值不能直接运算,所以报错提示。

解决办法:将数值类型转换为文本类型(Text. From(3)),然后与文本拼接。

在 Excel 中,日期函数或日期文本与数值可直接相加。表达式=DATE(2021,8,21)+3 或表达式 ="2021/8/21"+3 返回的值为 2021/8/24,而在 Power Query 中,表达式=#date(2021,8,21)+3 返回的错误提示如下:

```
Expression.Error: 无法将运算符 + 应用于类型 Date 和 Number。
详细信息:
    Operator = +
    Left = 2021/8/21
    Right = 3
```

出错原因:日期类型与数值类型不能直接相加。

解决办法:=#date(2021,8,21)+ #duration(3,0,0,0)。

在 Excel 中，Date()函数中的 year、month、day 3 个参数中，参数的数据类型为数值或数值文本都不会影响返回的值。表达式＝DATE(2021,8,"21")返回的值为 2021/8/21，而在 Power Query 中，表达式＝#date(2021,8,"21")返回的错误提示如下：

```
Expression.Error: 无法将值 "21" 转换为类型 Number。
详细信息:
    Value = 21
    Type = [Type]
```

出错原因：#date()函数内的 3 个参数的数据类型都必须是数值类型，不可以为文本或数值型文本。

解决办法：将数值型文本转换为文本或直接用数值，如＝#date(2021,8,Number.From("21"))。

在 Excel 中，当必选参数不足时，会弹出警示框。例如，在单元格输入表达式＝DATE(2021,8.21)后会弹出警示框，如图 3-9 所示。

图 3-9　报错提示

在 Power Query 中，表达式＝#date(2021,08.21) 返回的错误提示如下：

```
Expression.Error: 2 参数传递到了一个函数,该函数应为 3。
详细信息:
    Pattern =
    Arguments = [List]
```

出错原因：该函数共 3 个必选参数，目前仅提供了两个。

解决办法：补全所缺的参数。

在 Excel 中的所有函数，无论采用全部大写、全部小写、大小写相结合的任一方式，均不影响返回的结果。表达式＝UPPER("excel")、表达式＝upper("excel") 或表达式＝uPpEr("excel")返回的值均为"EXCEL"，而在 Power Query 中，表达式＝Text.upper("excel")返回的错误提示如下：

```
Expression.Error: 无法识别名称"Text.upper"。需要确保其拼写正确。
```

出错原因：Text.upper() 中的 upper 首字母未大写。

解决办法：正确书写 Text.Upper()。

在 Power Query 中，当函数名的大小写书写不正确时会报错，单词拼写不正确时也会报错。例如，表达式＝Text.Uppers("excel")返回的错误提示如下：

```
Expression.Error: 无法识别名称"Text.Uppers"。需要确保其拼写正确。
```

出错原因：Text.Uppers()中多了一个字母 s。

解决办法：正确书写 Text.Upper()。

在 Excel 中，find()函数的语法为 FIND(find_text, within_text, [start_num])。前两个参数为必选参数，如果在单元格内输入 Find("excel")，则会报错，如图 3-9 所示。

而在 Power Query 中，Text.PositionOf 函数的语法为 Text.PositionOf(text, substring, optional occurrence, optional compare)。前两个参数为必选参数，输入表达式＝Text.PositionOf("excel")返回的错误提示如下：

```
Expression.Error: 1 参数传递到了一个函数,该函数应介于 2 和 4 之间。
详细信息:
    Pattern =
    Arguments = [List]
```

出错的原因：Text.PositionOf()的参数介于 2 和 4 之间(共 4 个参数，其中有两个是必选参数)，目前仅输入了 1 个参数。

解决办法：补全参数，符合最少输入两个参数的要求。

尽管 Text.PositionOf()函数返回的值为 Number，但其函数的类别不是 Number。如果不小心将 Text.PositionOf()函数写成 Number.PositionOf()。例如，表达式＝Number.PositionOf("excel","e")，返回的错误提示如下：

```
Expression.Error: 无法识别名称"Number.PositionOf"。需要确保其拼写正确。
```

对于可选参数，特别是默认可选参数，写与不写都不影响返回的值。例如，表达式＝Text.PositionOf("excel","e",0)与表达式＝Text.PositionOf("excel","e")返回的值是完全一致的。

另外，Excel 中的 Find()函数与 M 语言中 Text.PositionOf()函数的参数中文本字符串的位置及查找后返回的值也略有所区别，如表 3-6 所示。

表 3-6　位置查找函数

函数类别	函　数	返回的值
Excel	FIND("e","excel")	1
M 语言	Text.PositionOf("excel","e")	0

2. 语法错误

在 Excel 中，对于任何类型的语法错误，系统都会预警提示。例如，在单元格输入 UPPER(("excel")，系统会弹出提示框，如图 3-10 所示。

单击“是(Y)”按钮，单元格返回值 EXCEL，而在 Power Query 中，表达式＝Text.Upper(("excel")返回的错误提示，如图 3-11 所示。

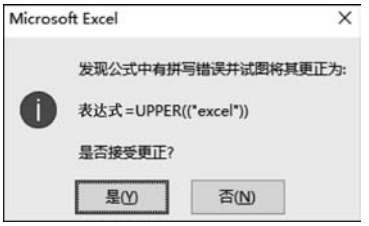

图 3-10　语法错误(1)

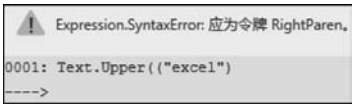

图 3-11　语法错误(2)

在图 3-11 中,RightParen 是右括号的意思,Paren 是 parenthesis(括号)的缩写。当出现语法报错时,读者一定要明白系统在具体指什么。

在 Excel 中,在单元格输入表达式=UPPER(("excel",),系统会弹出提示框,如图 3-12 所示。

在 Power Query 中,表达式=Text.Upper(("excel",)返回的错误提示如图 3-13 所示。

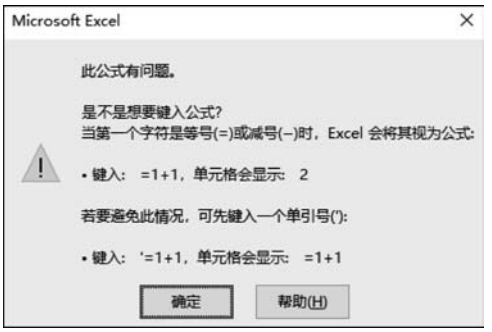

图 3-12　语法错误(3)

图 3-13　语法错误(4)

同理,表达式=Text.Upper("excel".)返回的错误提示如图 3-14 所示。

在图 3-14 中,Comma 是逗号的意思,"^"符号所指的位置即错误所在的位置。

再举例,表达式=Text.Upper("excel"]返回的错误提示如图 3-15 所示。

同理,在图 3-15 中,"^"符号所指的位置即错误所在的位置。通过以上几例的对比不难发现:"^"符号所指的位置一般为错误所在的位置,但提示语"应为令牌 Comma"则未必准确。

当文本函数中最基本的""(双引号)未成对出现时,则会报错"文字无效",相关文本内容都被"^"符号标识。例如,表达式=Text.Upper("excel])的报错如图 3-16 所示。

图 3-14　语法错误(5)

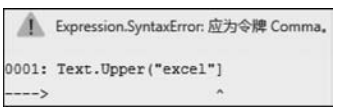

图 3-15　语法错误(6)

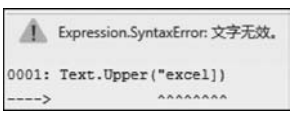

图 3-16　语法错误(7)

3. 结论推导

M 语言函数和 Excel 函数公式的主要共同点如下：

括号成对。对于存在多个参数的函数，很多函数有必选参数与可选参数；可选参数可以省略，但必选参数不可缺少；必选参数一旦不足，系统马上报错提示。

M 语言函数和 Excel 函数公式的主要区别如下：

M 语言函数对大小写敏感且第 1 个字母都是大写的，Excel 对大小写不敏感；M 语言函数对数据类型有严格的要求，Excel 中数据转换是隐式的；PQ 行号以 0 为基数，Excel 行号以 1 为基数。

3.2.3 函数及语法备忘

M 语言中函数的数量实在太多，并且有严格的大小写要求。很多情况下，忘记函数的拼写或语法是常有的事，找到一套备忘的方法同样很重要。

1. 借助图形化界面

在刚接触 M 语言的过程中，一次性掌握或记住所有函数及语法是有难度的。在想好解题思路的前提下，可以借助 Power Query 编辑器的图形化界面来降低学习的难度。例如，如果打算对某一列或某几列进行转换，则可先选中需转换的列，然后单击“转换”→“格式”→“小写”，最后对编辑栏生成的代码进行修改，如图 3-17 所示。

图 3-17 列表转换

接下来只需对编辑栏中的代码更改。例如，将编辑栏中的代码更改如下：

```
= Table.TransformColumns(源,{{"城市", each Text.Split(_,"、"){0}}})
```

2. 借助 # shared

如果对某个函数只有个模糊的记忆，则完全可以在编辑栏中输入“= # shared”，然后将 Record 转换为表，最后在表中进行查询即可。以上前两个步骤可合并为一个嵌套语句，代码如下：

```
= Record.ToTable(#shared)
```

如果所需查询的函数中含有单词 Transform，则接下来可以在 Power Query 编辑器中采用图形化界面完成操作。操作步骤如下：单击 Name 列右侧的“下拉”按钮（倒三角符号），选择下拉菜单中“文本筛选器”的“包含”，在弹出的“筛选行”对话框中，填入包含的值 Transform，单击“确定”按钮，如图 3-18 所示。

图 3-18　筛选行

在编辑栏显示的代码如下：

```
= Table.SelectRows(源, each Text.Contains([Name], "Transform"))
```

在编辑区显示的数据如图 3-19 所示。

	Name	Value
1	Table.TransformRows	Function
2	List.Transform	Function
3	List.TransformMany	Function
4	Record.TransformFields	Function
5	BinaryFormat.Transform	Function
6	Cube.Transform	Function
7	Table.TransformColumnNames	Function
8	Table.TransformColumns	Function
9	Table.TransformColumnTypes	Function

图 3-19　筛选的行

3. 借助编辑栏

通过图 3-19 的筛选，如果确定需寻找的函数为 Table.TransformColumns()，则可以在编辑栏直接输入表达式＝Table.TransformColumns 查找该函数的语法。语法查询结果如图 3-20 所示。

注意：在编辑栏查找某函数的语法时，函数的后面不能加括号。

图 3-20 语法查询

3.3 M 语言词法

以下内容为 M 语言最重要的基础知识,很重要但也很枯燥。希望读者能静下心来,多动手、多记忆、多练习,并举一反三。

3.3.1 值

在 M 语言中,值是通过计算表达式所生成的数据。值有原始值(Primitive Values,或称基元值)和结构型值(Structured Values)之分。举例:原始值(1,true, 3.1415,"abc");结构化值({1,2,3},{[A=1],{1,2,3}}, [a=1,b=a+1,c=a+b+2])。当值为文本时,需加双引号;当值为数值时,不需加引号。

注意:尽管许多值可以按字面形式写成表达式(例如,let A=1 in A,表达式 1 的计算结果为值 1),但值不是表达式。因为表达式是计算的方法,值是计算的结果。这种区别很细微,但很重要。

1. 值的种类

以下是 M 语言对值的分类,如表 3-7 所示。

表 3-7 值的分类

种 类	英 文	应 用 举 例
Null	Null	null, 1
逻辑	Logical	true false

续表

种　类	英　文	应用举例
数字	Number	0 1 −1 1.5 2.3e−5
时间	Time	#time(09,15,00)
日期	Date	#date(2013,02,26)
日期时间	DateTime	#datetime(2013,02,26, 09,15,00)
日期时区时间	DateTimeZone	#datetimezone(2013,02,26, 09,15,00, 09,00)
持续时间	Duration	#duration(0,1,30,0)
文本	Text	"hello"
二进制	Binary	#binary("AQID")
列表	List	{1, 2, 3}
记录	Record	[A = 1, B = 2]
表格	Table	#table({"X","Y"},{{0,1},{1,0}})
函数	Function	(x) => x + 1
类型	Type	type { number } type table [A = any, B = text]

2. 运算符

在 M 语言中,主要有以下运算符:

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

在 M 语言中:表为 Table,每行的内容为一个 Record,每列的内容为一个 List;行标用大括号{ },初始值为 0;列标用中括号[ ],[]内为字段名,列标名不用加引号。例如,源{0}[Data],用于获取源表中第一行 Data 列的内容,其中{}、[]为表级运算符。

注意:运算符是有优先级的。例如,*/(乘除)优先于+−(加减)。

3. 值的运算

M 语言是强类型的,不同类型的值之间不可以进行运算;相同类型的值之间并非可以进行各类运算(例如,对比运算、算术运算、逻辑运算等),各类相同数据类型间可进行的运算如表 3-8 所示。

表 3-8　常用表达式及可进行的运算

表达式及运算符				(M 语言中常用)数据类型												
常用表达式的种类	表达式	运算符	中文含义	null	逻辑	数字	时间	日期	日期时间	日期时区时间	持续时间	文本	二进制	列表	记录	表格
相等表达式	x=y	=	等于	√	√	√	√	√	√	√	√	√	√	√	√	√
	x<>y	<>	不等于	√	√	√	√	√	√	√	√	√	√	√	√	√

续表

表达式及运算符				(M语言中常用)数据类型												
常用表达式的种类	表达式	运算符	中文含义	null	逻辑	数字	时间	日期	日期时间	日期时区时间	持续时间	文本	二进制	列表	记录	表格
关系表达式	x>=y	>=	大于或等于	√	√	√	√	√	√	√	√	√	√			
	x>y	>	大于	√	√	√	√	√	√	√	√	√	√			
	x<y	<	小于	√	√	√	√	√	√	√	√	√	√			
	x<=y	<=	小于或等于	√	√	√	√	√	√	√	√	√	√			
算术表达式	x & y	&	组合	√			√	√				√		√	√	√
	x+y	+	加	√		√	√	√	√	√	√					
	x-y	-	减	√		√	√	√	√	√	√					
	x * y	*	乘	√		√					√					
	x/y	/	除	√		√					√					
逻辑表达式	x and y	and	条件逻辑与	√	√											
	x or y	or	条件逻辑或	√	√											
	not x	not	逻辑非	√	√											

如表3-8所示,数值间是不可以进行组合(&)运算的。表达式= 1 & 2返回的错误提示如下:

```
Expression.Error: 无法将运算符 & 应用于类型 Number 和 Number。
详细信息:
    Operator = &
    Left = 1
    Right = 2
```

如表3-8所示,列表间是可以进行比较运算(=、<>、>、>=、<、<=)的。表达式= {1,2,3}<>{1,3,2}返回的值为true。

3.3.2 变量

变量是对值的命名。在let…in…语句结构中,变量是步骤名称,当变量中存在空格时,应对字段加引号,然后在引号前面加#(井号),例如,#"A B"。

在Record结构中,变量是字段名,即使变量中存在空格并能正常使用,即无须对字段加引号(及在引号前加#);当然,如果加上引号及#号也不会报错,例如,[A B=1]与[#"A B"=1]都是允许的,并且二者是等效的。

3.3.3 环境

环境是由let…in…或Record结构中的所有变量组成。在每个let…in…结构中,由所有变

量组成一个 let…in…环境；在每个 Record 结构中，由所有字段名（变量）组成 Record 环境。

在同一环境中，每个变量都必须是唯一的，所以变量也可以称为“唯一标识符”或“标识符”。如果打算在同一环境中命名两个相同的变量（或称应用步骤的名称、标识符），则系统会报错提示。例如，表达式= let A=1,A=2 in A 或表达式 = [A=1,A=2]的错误提示均为

```
Expression.Error: 名称"A"被定义多次。
```

在 let…in…结构中，由所有变量构成了环境，let 表达式使用包含所有变量的环境来计算后面的表达式。应用举例，代码如下：

```
let x = 1,y = x + 3,z = x + y in z
```

返回的值为 5。代码中，x 的环境为 y、z；y 的环境为 x、z；z 的环境为 x、z。

在 Record 结构中，由所有字段构成了环境，初始字段表达式使用修改后的环境计算每个字段的子表达式。应用举例，代码如下：

```
[x = 1,y = x + 3,z = x + y]
```

返回的值为 Record。代码中，x 的环境为 y、z；y 的环境为 x、z；z 的环境为 x、z。

在 let…in…结构中嵌套 Record 结构或在 Record 结构中嵌套 let…in…结构也是允许的。let 结构应用举例，代码如下：

```
//ch302 - 002
let
  x = 1,
  y = [x = 1,y = x + 3,z = 5][y],
  z = x + y
in
  z
```

Record 结构应用举例，代码如下：

```
[x = 1,y = let x = 1,y = x + 3 in y, z = x + y][z]
```

以上代码返回的值均为 5。在以上两个嵌套代码中，每个代码中的 let 表达式中的 y 与 Record 中的字段 y 的所处环境均不相同，所以不会产生标识符或变量冲突，但不易于新手理解或未来的代码维护，所以不建议这样命名。以此 Record 结构为例，修改后的代码如下：

```
[x = 1,y = let a = 1,b = a + 3 in b, z = x + y][z]
```

以上代码相比[x=1,y=let x=1,y=x+3 in y, z=x+y][z]更易于理解。

在实际代码编写过程中，当运算过程中存在某个或某些复杂的变量重复调用时，采用let…in…结构嵌套let…in…或Record结构是一种高效的处理方式，并且代码易于识别、易于修改。

3.3.4 令牌

在M语言中，令牌(token)是指标识符(identifier)、关键字(keyword)、文字(literal)、运算符(operator)或标点符号(punctuator)，但用于分隔标记的空白和注释不是令牌。

3.3.5 标识符

在M语言中，标识符有通用化标识符(例如，let…in…语句中的步骤名称，record结构中的字段名称)和带引号的标识符(例如，let…in…语句中存在空格号的步骤名称，需对其加上#"")。

在M语言中，关键字是保留的类似标识符的字符序列，不能直接用作常规标识符，但可以用作带引号的标识符。

注意：在M语言的Record结构中，当字段名称中存在关键字或空白时，不必使用带引号的标识符，可以直接用常规标识符。

3.3.6 关键字

在M语言中，以下是系统内置的关键字。

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

所有关键字都必须小写，部分关键字前面需加上转义字符(#)。以is和as关键字为例，对关键字的作用进行简单说明，is运算符用于确定值的类型是否与给定类型兼容；as运算符用于检查该值是否与给定类型兼容，如果不兼容则会引发错误。否则将返回原始值。

以as关键字指定数据类型，代码如下：

```
= let AB = (A as number, B as text) => Text.From(A)&B in AB
```

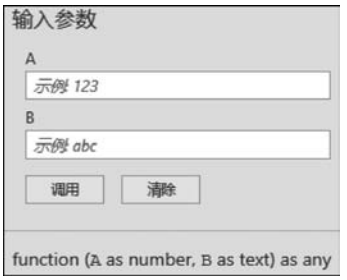

图3-21 输入参数

运行此代码，返回的结果如图3-21所示。

注意：is和as运算符仅接受初始类型(Primitive)作为其正确的操作数。M语言不提供用于检查值是否符合自定义类型的方法。

另外，M语言的关键字里面没有for、while这样的关键字。在M语言中，实现循环是通过特定的函数或者运算符实现的，按照实现原理的不同，可以初步地分为遍历、迭代与递归三大类，例如，List.Transform()、List.TransformMany()、List.Accumulate()、List.Generate()等。

3.3.7 标点符号

标点符号用于分组和分隔。在M语言中，常见的标点符号的中英文对照如表3-9所示。

表3-9 常用标点符号

符 号	英文提示	中文含义
[ ]	Bracket	方括号
:	Colon	冒号
,	Comma	逗号
=	Equals sign	等号
()	Paren	圆括号
+	Plus	加号
""	Quote	引号
;	Semi-colon	分号
	Space	空隔号

3.3.8 空白分隔符

空格(空白分隔符)用于分隔M语言中的注释和令牌。空格包括空格字符(它是Unicode字符类的一部分)及水平和垂直制表符(#(tab))、换行符序列(包括回车符#(cr)、换行符#(lf)、后跟换行符的回车符#(lf,cr))等。制表符#(tab)、回车符#(cr)、换行符#(lf)等是较常用的字符转义序列。

转义序列也可以包含短(4个十六进制数字)或长(8个十六进制数字)Unicode码位值。例如，以下3个转义序列是等效的：#(000D)短(4个十六进制数字)、#(0000000D)长(8个十六进制数字)Unicode码位值、#(cr)字符转义序列。

单个转义序列中可以包含多个转义码，它们之间用逗号分隔。例如，#(cr)#(lf)和#(cr,lf)这两个序列是等效的。

3.4 M语言表达式

表达式是由运算符和运算对象组成的，它用于构造值的公式。单独的一个运算对象(常量、变量或算术)也可作为表达式，它是最简单的一种表达式。常见的表达式有逻辑表达式、算术表达式、文本字符串表达式。

表达式可以是简单的表达式，也可以是复杂的表达式，复杂的表达式是建立在众多子表式的基础之上的。例如，[A=3, B={2}, C={if A>2 then 3 else null}&B]是父表达式，而3、{2}、{if A>2 then 3 else null}&B是子表达式。

3.4.1 表达式

M语言中的各类表达式如表3-10所示。

表3-10 M语言的表达式

序 号	表 达 式	英 文 描 述	运算符或关键字
1	逻辑表达式	logical-or-expression	and、or、is、as、相等、关系、+、-、*、/、一元表达式(+、-、not)
2	if表达式	if-expression	if…then…else…
3	let表达式	let-expression	let…in…
4	each表达式	each-expression	each _
5	函数表达式	function-expression	fx=(x,y)=>
6	主表达式	primary-expression	如表3-11所示
7	报错表达式	error-raising-expression	error expression
8	错误处理表达式	error-handling-expression	try…otherwise…

在M语言中,常用主表达式如表3-11所示。

表3-11 主表达式

序 号	主 表 达 式	英 文 描 述	主 运 算 符
1	列表表达式	list-expression	{x,y,…}
2	项访问表达式	item-access-expression	x{y}
3	记录表达式	record-expression	[i=x,…]
4	字段访问表达式	field-access-expression	x[i]
5	标识符表达式	identifier-expression	i、@i
6	带圆括号表达式	parenthesized-expression	(x)
7	调用表达式	invoke-expression	x(…)
8	文本表达式	literal-expression	"A123"

3.4.2 逻辑表达式

1. and、or

and和or是较常用的逻辑运算符。and运算符在两个条件都满足时,返回值为true;只有一个条件满足时,返回值为false。应用举例,代码如下:

```
= 2 > 3 and 5 > 4          //false
= 2 > 3 or 5 > 4           //true
```

or运算符在两个条件中只要一个条件满足时就返回值true;如果两个条件都不满足,则返回值为false。应用举例,代码如下:

```
= 2 > 3 or 5 > 4        //true
= 2 > 3 or 4 > 5        //false
```

如果条件参数结果为 true,则返回值为 false。同理,如果条件参数结果为 false,则返回值为 true。应用举例,代码如下:

```
= not false        //true
= not true         //false
```

2. is、as

is 运算符并不真正执行转换,它只是检查指定的对象与给定的类型是否兼容。应用举例,判断数值 3 是否为文本值,代码如下:

```
= 3 is text
```

返回的值为 false。在 M 语言中,函数中包含 Is 的均为信息类函数,返回值为 true 或 false。

as 运算符只适用于引用类型或可以为 null 的类型,而无法执行其他的转换。应用举例,将自定义函数 fn 的 a 参数的数据类型定义为数值,代码如下:

```
= let fn = (a as number) => a in fn
```

当 a 值不是指定数据类型(数值类型)时,参数将无法被调用。

3. 相等运算符

比较运算符可用于相等(=、<>)或关系(>、>=、<、<=)的比较。以下是一些相等比较,代码如下:

```
= 1 = 1,                  //true
= 1.0 = 1                 //true
= "a" <> 2                //true
= 2 = 1                   //false
= #nan = #nan             //false
= #nan <> #nan            //true
```

返回的值为 true 或 false。

参照表 3-9,列表间比较是允许的。只要项相等、顺序相同,返回值就为 true,代码如下:

```
= {1,2} = {1,2}          //true
= {2,1} = {1,2}          //false
= {1,2,3} = {1,2}        //false
```

参照表 3-9，对记录进行比较也是允许的。只有当字段数相同、字段名称相同时，返回值才为 true，代码如下：

```
= [A = 1,B = 2] = [A = 1,B = 2]            //true
= [B = 2,A = 1] = [A = 1,B = 2]            //true
= [A = 1,B = 2,C = 3] = [A = 1,B = 2]      //false
= [A = 1] = [A = 1,B = 2]                  //false
```

参照表 3-9，表与表之间的比较也是允许的。应用举例，代码如下：

```
= #table({"A","B"},{{1,2},{3,4}}) =  #table({"B","A"},{{4,3},{2,1} })
```

返回的值为 false。

如果列的顺序不同，但行列值能一一对应，则返回的值为 true，代码如下：

```
= #table({"A","B"},{{1,2},{3,4}}) =  #table({"B","A"},{{2,1}, {4,3}})
```

4. 关系表达式

关系表达式(>、>=、<、<=)返回的值为 true 或 false，代码如下：

```
= "a" > "A"          //true
= "c" > "a"          //true
= "a" > "一"         //false
= "ab" >= "ac"       //false
= 3 >= 2             //true
= true > false       //true
= true < false       //false
```

3.4.3 if 表达式

if 表达式的语法结构如下：

```
if if - condition then true - expression else false - expression
```

if…then…else…的简单应用举例，代码如下：

```
= if 5 > 4 then "ok" else "error"      //ok
```

if…then…else…的多条件应用举例，代码如下：

```
=  if 5 > 12 then "ok" else if 5 > 6 then "ok" else "no"      //no
```

在刚接触 M 语言的多条件应用时，如果对语法不熟悉，则可以通过 Power Query 的图

形化界面来操作。单击“添加列”→“条件列”，弹出的窗口如图 3-22 所示。

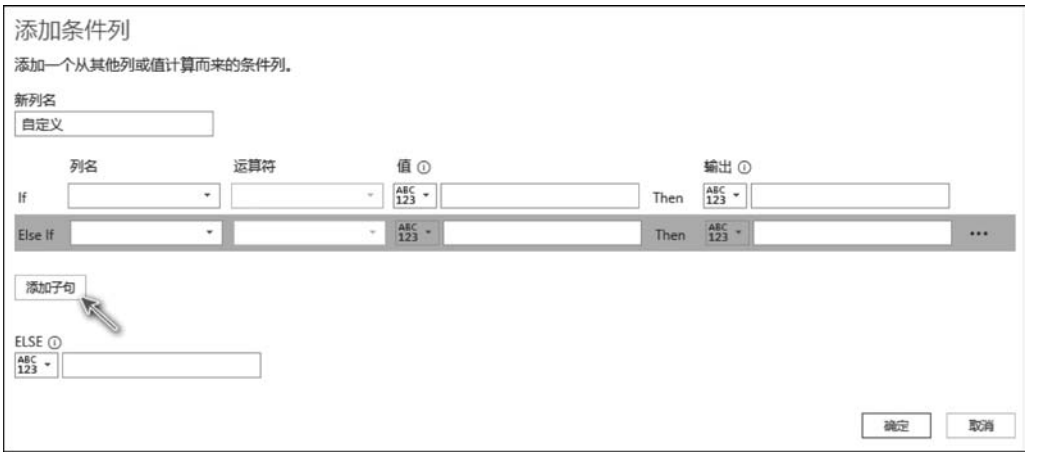

图 3-22　添加条件列

当需要添加多条件时，可通过单击“添加子句”获得条件行，在“Else IF、Then”中选择“列名、运算符”然后输入“值、输出”实现。

3.4.4　let 表达式

let…in…语句结构用于创建一个多步骤的综合查询，每个综合查询都以 let 开头、以 in 结尾；let、in 是 M 语言中的关键字，只能是小写。在 let..in..结构中，let 用于计算，并对结果命名；in 用于显示结果。

let 之后所连接的每个步骤都有一个步骤名称，称为“标识符”“步骤名称”或“变量”。标识符用于引用值的名称，当标识符（步骤名称、变量）中存在空格时，需要用 # 标识符来包含空格（名称在引号中，例如，#"A B"，也可称为带引号的标识符）；不带空格的标识符称为常规标识符（例如，A、B）。

在 let…in…结构中，in 之前不能有逗号，其他的每个步骤都以逗号结尾；in 后面所接的是语句的输出，代码举例如下：

```
//ch304 - 010
let
    源 = 2,
    #"A B" = 源 + 3 * (源 + 1) //#"A B",用#标识符来包含空格(名称在引号中)
in
    #"A B"
```

在以上代码中，源、#"A B"是对标识符的直接调用；//是注释符，用于单行注释，//后面的语句不执行任何操作。以上代码返回的值为 11。

在 let…in…结构中，in 表达式返回的值可以是标识符（let…in…中的任一步骤），也可

以是标识符调用的结果(例如,A+B),代码如下:

```
//ch304-011
let
    A = 1,
    B = 2
in
    A+B
```

在以上代码中,A 和 B 为标识符(或称步骤名称、变量),A+B 为标识符(或称步骤名称、变量)的调用。返回的值为 3。

在 let…in…结构中,如果 in 的前面存在逗号(,),系统则会报错如下:

```
Expression.SyntaxError: 逗号不能位于 In 之前。
```

以 M 语言中,/*……*/为多行注释,在"/*"与"*/"之间所编写的任何内容都不会被运行。在 let…in…结构中,任何形式对标识符的调用其原理都是一样的,代码如下:

```
//ch304-012
let
    A = 1,
    B = 2,
    C = A+B
/* 语法注释:
(1)  A、B、C、A+B+C 是标识符,也可称为"变量""步骤名称"。
(2)  A = 1,1 是表达式,A 是变量; 变量是对值的命名。
(3)  C=A+B, A+B 是表达式,A 和 B 是子表达式,C 是变量。
(4)  在 let…in…语句中,所有的这些变量构成了 let 表达式的环境。
(5)  环境中的每个变量在环境中都有一个唯一的名称,称为标识符。
(6)  如果尝试在同一环境中定义两个相同的变量,则系统会自动报错。
*/
in
    A+B+C
```

以上代码返回的值为 6。

在 let…in…结构中,某 let…in…结构可以将整个过程当成一个步骤进行嵌套,代码如下:

```
//ch304-013
let C=
    let
     A = 1,
     B = 2
    in
     A+B
in C
```

返回的值为3。

在let…in…结构中，当某复杂变量需要在其他变量中反复调用时，相比let…in…中let…in…的嵌套，在let…in…结构中嵌套record结构是一种高效的处理方式，值得花时间去了解与掌握。

3.4.5　each表达式

each表达式是M语言中的一种简写形式，声明一个名为_（下画线）的单形式参数的无类型函数，通常用于提高函数的可读性；当_被调用时，什么类型的参数传递给了它，那么它就代表什么数据类型。在实际使用过程中，当多个each _被嵌套使用在同一表达式中时，为避免上下文冲突，有时会采用(x)=> x等形式来取代each _。

each是M语言中的关键字（只能小写），代码如下：

```
//ch304-014
let
    源 = #table({"a","b"},{{1,3},{2,6}}),
    A =  Table.AddColumn(源, "EACH", each _ ),
    B = Table.AddColumn(A, "小计", each [a]+[b])
in
    B
```

在M语言中，字段前面的下画线是可以省略的。以上代码中each [a]+[b]相当于each _[a]+_[b]，返回的值如图3-23所示。

	a	b	EACH	小计
1	1	3	Record	4
2	2	6	Record	8

图3-23　添加列

注意：List中的下画线不能省略。

3.4.6　函数表达式

在M语言中，函数主要有以下几种：

（1）内置函数，例如，Text.From()，它是系统自带的标准库函数。

（2）自定义函数，自定义函数的基本语法为函数名=（参数1，参数2，参数3...）=>表达式。例如，= let fn = (x,y)=> x+y in fn，参数与表达式之间用=>隔开，这是固定组合。

（3）参数函数，即函数内参数的类型为function，function的中文意思是“函数”。例如，函数List.Generate()的语法如下：

```
List.Generate(
    initial as function,
    condition as function,
    next as function,
    optional selector asnullable function
) as list
```

List.Generate()函数内的4个参数类型均为function。

3.4.7 主表达式

1. 列表表达式

“列表”值是值的有序序列,代码如下:

```
= {1..3}                                                    //{1,2,3}
= List.Sum({1..3})                                          //6
= List.Transform({1..3}, each _ * 3)                        //{3,6,9}
= List.Select(List.Transform({1..3}, each _ * 3), each _ > 4)   //{6,9}
```

列表表达式经常被放置于let…in…结构中。在以下代码中,标识符右侧的(=……)均为列表表达式,代码如下:

```
//ch304 - 016
let
    A = {1..3},
    B = {2..4},
    C = {List.Sum(A)}&List.Select(B,each _ > 3)
in
    A&B&C
```

返回的值为{1,2,3,2,3,4,64}。

2. 项访问表达式

项访问(item-access)是通过“位置索引运算符”({ })按其数字索引访问列表中的项目,从列表或表中选择对应的值。“深化”是“项访问”的通俗说法,位置索引是从0开始的。从列表中深化第1个值,代码如下:

```
= {1,3}{0}                          //从列表中深化第1个值
```

返回的值为1。

如果索引号大于列表长度(列表中的元素的个数),则会返回错误,代码如下:

```
= {1,3}{3}
```

返回的错误提示如下:

```
Expression.Error: 枚举中没有足够的元素来完成该操作。
详细信息:
    [List]
```

对$x\{y\}$项访问求值时,为避免因为x或y的各类原因而引起的表达式错误

(Expression.Error),可采用 $x\{y\}$? 形式,将列表或表中 x 不存在的位置(或匹配项)y 而引起的错误值转换为 null 值;当然,如果存在多个 y 匹配项,仍会导致错误。$x\{y\}$? 项访问应用举例,代码如下:

```
= {1,3}{3}?                    //null
= {"城市","排名","得分"}{0}?    //"城市"
= {1,[排名 = 2],3}{1}?         //[排名 = 2]
= {true,false}{2}?             //null
```

在 M 语言中,值有原始值(Primitive Value,或称"基元值")与结构化值(Structured Values)之分,列表中允许结构化值(列表、记录、表格)存在。从列表中选择对应的值,并且在列表中嵌套 Record 数据结构,代码如下:

```
= {{1..6},3}{0}          //从列表中深化第 1 个值
```

结果为{1,2,3,4,5,6}。如果需要深化该列表中的第 3 个值也是可以的,代码如下:

```
= {{1..6},3}{0}{2}          //多层深化
```

返回的值为 3。

在 Excel 中,通过"数据→新建查询→从文件→从工作簿"获取 AR005.xlsx 工作簿,在弹出的"导航器"中选择"转换数据",获取的数据如图 3-24 所示。

	Name	Data	Item	Kind	Hidden
1	2019_5	Table	2019_5	Sheet	FALSE
2	2019_9	Table	2019_9	Sheet	FALSE
3	2020_3	Table	2020_3	Sheet	FALSE
4	2020_4	Table	2020_4	Sheet	FALSE
5	2020_5	Table	2020_5	Sheet	FALSE
6	2020_7	Table	2020_7	Sheet	FALSE
7	2020_9	Table	2020_9	Sheet	FALSE
8	2019_10	Table	2019_10	Sheet	FALSE
9	2019_11	Table	2019_11	Sheet	FALSE

图 3-24　获取数据

从表中选择对应的值(例如,[Name]= "2020_3"),代码如下:

```
= 源{2}          //源,查询引用
```

通过项访问后返回的值为 Record,如图 3-25 所示。

表可以来自于工作簿,也可以来自于手动创建的表。对于手动创建的表的项访问也是可以的,代码如下:

图 3-25　深化引用

```
= #table(
        {"城市","排名","得分"},
        { {"北京",1,95},
          {"上海",2,93}
         }
    ){0}
```

关于表的创建原理可参见第 9 章。以上代码返回的值如图 3-26 所示。

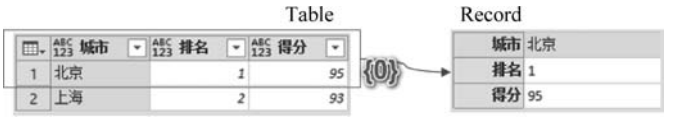

图 3-26　行的深化引用

3. 记录表达式

“记录”是一组字段。字段是名称/值对，其中名称是字段记录中唯一的文本值。记录值的文本语法允许将名称写成不带引号的形式，这种形式称为“标识符”。

在以下代码中，A、B 是记录中的字段，也可称为记录的标识符。应用举例，代码如下：

```
[
     A = 1,          //A 字段
     B = 2 + 3       //B 字段
]
```

返回的值为 Record。

在下面的代码中，字段 B 存在对字段 A 的引用，代码如下：

```
[
     A = 1,
     B = A + 3
]
```

返回的值为 Record。假如 A 是一个复杂的表达式，采用以上 Record 结构的写法可以瞬间让代码变得简洁。

在 M 语言中，变量间依据依赖关系进行计算，应用举例：

```
[
      A = B * 2,
      B = C + 3,
      C = 1
]
```

返回的值为 Record，与大多数读者的习惯写法[A＝1，B＝A＋3，C＝B＊2]的返回值相同。

在 Record 记录中，字段字符间可以存在空格，但不能存在运算符。对于存在运算符的字段，必须采用引用标识符的方式，代码如下：

```
[
   #"A + B" = A + B,
   A = 1,
   B = 2
]
```

返回的值为 Record。

字段名称相同的两个记录是允许连接的。连接之后的结果为新值替换旧值，代码如下：

```
[ A  = 1 ] & [A = 2]
```

返回的值为[A＝2]，其遵循的是数据处理过程中“无则新增、有则更改”的原则。

字段名称不同的两个记录也是允许连接的。连接的结果相当于记录的追加，代码如下：

```
[ A  = 1 ] & [B = 2]
```

返回的值为字段追加后的 Record。

记录中嵌套记录或其他数据结构(列表、表格)都是允许的，代码如下：

```
[
A = [x = 1, y = 2, z = x + y],
B = 2
]
```

返回的值为存在嵌套 Record 的 Record。

4. 字段访问表达式

字段访问是对某一列的深化。使用运算符 x[y] 按字段名称在记录中查找字段，代码如下：

```
[A = 1,B = 2][B]        //2
[A = 1,B = 2][C]        //error
[A = 1,B = 2][C]?       //null
```

通过对记录中对应字段的访问返回字段对应的值；计数从 0 开始，如果索引号大于列表长度(列表中元素的个数)，则会返回错误；如果不想返回错误，则可采用 x{y}? 形式，将返回的错误值转换为 null 值。

运算符支持对多个字段进行集体访问，代码如下：

```
[A = 1,B = 2][[B]]      //[B = 2]
[A = 1,B = 2][[C]]      //error
[A = 1,B = 2][[B],[C]]? //[B = 2,C = null]
```

在 let…in…结构中，当某变量存在反复调用时，将其先定义好再反复调用不失是一个好方法。相比 let…in…嵌套的方式，采用 Record 结构将会更为高效，代码如下：

```
[
     A = [j = 11,k = 1.1],
     B = A[j] + A[k]
][B]
```

返回的值为 12.1。此用法使用频率较高，需重点掌握。

采用 Record 结构作为 let 表达式中的变量，然后供其他步骤的深化调用，代码如下：

```
//ch304 - 029
let
    A = [y = 1,m = 2,d = 3],
    B = [j = 2,k = 3,l = 4]
in
    A[y] + B[j]
```

返回的值为 3。

继续举例 Record 结构供其他步骤的深化调用，代码如下：

```
//ch304 - 030
let
    A = [x = 1, y = 2, z = x + y],
    B = 2
in
    A[z] + B
```

返回的值为 5。

为了形成子表达式的环境，新变量会与父环境中的变量进行“合并”，代码如下：

```
[
A = [a=1,b=3,c = a+b],
B = 2,
C = A[c]+B
][C]
```

返回的值为6。

对于列表型嵌套表达式，先对行索列深化引用再对列字段进行深化是允许的，代码如下：

```
{[a=1],3}{0}[a]
```

返回的值为1。

对于记录嵌套表达式，先用字段对其他字段的值进行列表深化和字段深化，最后对记录的最后一个字段进行深化，代码如下：

```
[A = { [a=1] ,[a=3]}, c= A{0}[a] + A{1}[a] ][c]
//[A = { [a=1] ,[b=3]}, c= A{0}[a] + A{1}[b] ][c]
```

列表索引值0和1分别代表列表中的第一项和第二项；在M语言中，系统默认的索引值都是从0开始的，找不到的索引值均显示为－1。以上代码的返回值为4。

记录中的字段表达式为复杂型表达式也是允许的，代码如下：

```
[
      A = { [a=1] ,
            [a=3]},
      c= if A{0}[a] > 2 then A{0}[a] else 0 + A{1}[a]
  ][c]
```

返回的值为3。

以上代码可以进行简化，代码如下：

```
[
      A = { [a=1] ,
            [a=3]},
      c = A{0}[a],
      d= if c > 2 then c else 0 + A{1}[a]
  ][d]
```

返回的值为3。

{[]}结构：用于获取指定列的条件所在行的整行记录。例如，{[城市＝"北京"]}是以下代码中{0}的具体化，代码如下：

```
= #table(
            {"城市","排名","得分"},
            { {"北京",1,95},
              {"上海",2,93}}
    ){[城市 = "北京"]}
```

返回的值为 Record。

更多举例，代码如下：

```
= #table({"城市","排名","得分"}, {{"北京",1,95},{"上海",2,93}}){[排名 = 1]}
= #table({"城市","排名","得分"}, {{"北京",1,95},{"上海",2,93}}){[排名 = 2]}
```

以上两个表达式返回的值均为 Record。

如果存在以下情形，则代码将报错提示，代码如下：

```
= #table({"城市","排名","得分"}, {{"北京",1,95},{"上海",2,93}}){[排名 = 3]}
//null,不存在 y

= #table({"城市","排名"}, {{"北京",1},{"上海",1}}){[排名 = 1]}
//error,存在多个 y
```

在查询中，直接字段访问用于获取表的整列或记录中字段的值，在项访问中嵌套的字段访问（获取指定列的条件所在行的整行记录）在表查询时可实现类似于 Excel 的 vlookup 功能，即“先项访问再列访问”或“先列访问再项访问”获取条件行与列交叉的记录。这些都是使用频率较高的，后续章节会继续举例说明。

5. 标识符表达式

标识符表达式用于引用环境中的变量，有两种方式：专属标识符引用、包含标识符引用。最简单的方式是专属标识符引用，即在其他步骤上直接输入标识符（变量）的名称，达到引用的目的。另一种方式是包含标识符引用，采用“@标识符”方式。

“包含标识符引用”应用举例，代码如下：

```
= [fn = (x) => if x <= 1 then 1 else x * @fn(x - 1),y = fn(5)][y]
```

代码中@是范围操作符，fn 是标识符，@fn 用于递归运算。返回的值为 120。

以上代码采用 let…in…结构也是允许的，代码如下：

```
//ch304 - 039
let
```

```
    fn = (x) => if x <= 1 then 1 else x * @fn(x - 1),
    y = fn(5)
in
    y
```

运行代码，返回的值也为 120。

6. 带圆括号表达式

当表达式中有用到 and 或 or 多条件判断时，可以带圆括号进行分组运算，通过圆括号分组及确定逻辑的优先级。应用举例，代码如下：

```
= ("a" is text and 3 > 2) or (Number.From("12") > 6 or Logical.From(2) = true)//true
```

带圆括号表达式可用于 if 表达式的条件中。应用举例，代码如下：

```
= if ("a" is text and 3 > 2) or (Number.From("12") > 6 or Logical.From(2) = true) then
"Excel2016" else "2016"  //Excel2016
```

或者用于四则运算中，用圆括号来确定计算的优先组。应用举例，代码如下：

```
= (2 + 3) * 4          //20
```

7. 调用表达式

“函数”是一个值，当带着参数进行调用时，将生成一个新值。应用举例，代码如下：

```
= let fn = (x, y) => (x + y) / 2 in fn
```

运行代码，结果如图 3-27 所示。

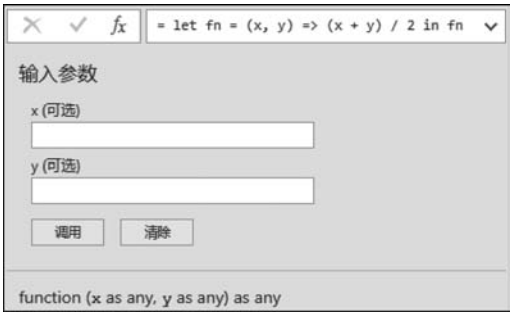

图 3-27　输入参数

如果在 x 与 y 中不输入任何值就单击“调用”按钮，则生成的值为 null；如果在 x 中输入 3，在 y 中输入 9，单击“调用”按钮，则生成的值为 6。同时在 Power Query 查询区会生成一个“调用的函数”，在编辑栏出现的代码为 = 查询 1(3, 9)，如图 3-28 所示。

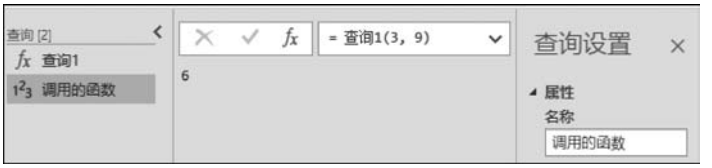

图 3-28 调用自定义函数

3.4.8 报错表达式

错误是由运算符和函数遇到错误条件或使用了错误表达式导致的，可以使用 try 表达式来处理错误，也可以用 error 指示错误发生的原因。案例应用，代码如下：

```
//ch304 - 042
let
    一 = [
            A = 1,
            B = 16,
            C = if B < 0 then error "错误提示：B 为负值" else A + B
        ],
    二 = try Number.ToText(一[C]),
    三 = "C 值：= " &
        (  if 二[HasError]
           then 二[Error][Message]
           else 二[Value]
         )
in
    三
```

以上代码返回的值为“C 值：= 17”。如果将变量“一”中的 B 值更改为－16，则返回的值为“C 值：= 错误提示：B 为负值”。

3.4.9 报错处理表达式

try…otherwise…是较为常用的一个语句结构，类似于 Excel 中的 iferror()函数。案例应用，代码如下：

```
let A = 12, B = "AB", C = try A + B otherwise null in C
```

在上述代码中，数值与文本是不能直接进行四则运算的，其结果一定会报错；通过 try 发现错误后，通过 otherwise 将值指定为 null，最终以上代码返回的值为 null。