Power Query M Functions

2017-2-19

- **Access.Database**

返回 Access 数据库 database 的结构表示形式。可以指定可选的记录参数 options 来控制以下选项:

CreateNavigationProperties :逻辑(true/false)，用于设置是否在返回的值上生成导航属性(默认值为 false)。

NavigationPropertyNameGenerator :函数，用于创建导航属性的名称。

例如，可以将记录参数指定为 [option1 = value1, option2 = value2...]。

```
Access.Database( database as binary, optional options as nullable record) as table
```

- **ActiveDirectory.Domains**

返回与指定域或当前计算机的域(如果未指定任何域)处于同一个林中的 Active Directory 域的列表。

```
ActiveDirectory.Domains(optional forestRootDomainName as nullable text) as table
```

- **AdoDotNet.DataSource**

使用提供程序名称 providerName 和连接字符串 connectionString 返回 ADO.NET 数据源的架构集合。 connectionString 可以是文本或记录的属性值对。属性值可以是文本或数字。

```
AdoDotNet.DataSource( providerName as text, connectionString as any) as table
```

- **AdoDotNet.Query**

返回通过使用 ADO.NET 提供程序 providerName 的连接字符串 connectionString 运行 query 的结果。 connectionString 可以是文本或属性值对记录。属性值可以是文本或数字。

```
AdoDotNet.Query( providerName as text, connectionString as any, query as text) as table
```

- **AnalysisServices.Database**

从服务器 server 上的 Analysis Services 数据库 database 中返回多维数据集或表格模型的表。可以指定一个可选记录参数 options 来控制以下选项:

区域性：指定数据的区域性的区域性名称。与 "Locale Identifier" 连接字符串属性相对应。

TypedMeasureColumns：一个逻辑值，指示多维或表格模型中指定的类型是否用于添加的度量值列的类型。当设置为 false 时，将为所有度量值列使用类型 number 。此选项的默认值为 false 。

查询：在服务器上运行并使用表格格式进行检索的文本。

AnalysisServices.Database(server as text, database as text, optional options as nullable record) as table

- **AnalysisServices.Databases**

返回关于 Analysis Services 实例 server 的数据库。可提供可选的记录参数 options 以指定附加属性。该记录可以包含以下字段:

Culture：指定数据区域性的区域性名称。这对应于“区域设置标识符”连接字符串属性。

TypedMeasureColumns：指示在多维模型或表格模型中指定的类型是否用于添加的度量值列类型的逻辑值。设置为 false 时，类型 number 将用于所有度量值列。此选项的默认值为 false 。

AnalysisServices.Databases(server as text, optional options as nullable record) as table

- **Any.Type**

表示所有值的类型。**AzureStorage.Blobs**

返回包含从 Azure 存储库的帐户 URL account 中找到的每个容器行的导航表。每行包含一个到容器 Blob 的链接。

AzureStorage.Blobs(account as text) as table

- **AzureStorage.Tables**

返回一个导航表，它对于在 Azure 存储库的帐户 URL account 上找到的每个表包含一行。

每一行都包含指向 Azure 表的链接。

AzureStorage.Tables(account as text) as table

● Binary.Buffer

缓冲内存中的二进制值。 此调用的结果是一个稳定的二进制值，这意味着它将具有确定的字节长度和顺序。

Binary.Buffer(binary as nullable binary) as nullable binary

创建二进制值的稳定版本。

Binary.Buffer(Binary.FromList({0..10}))
#binary({0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10})

● Binary.Combine

将一组二进制值合并成单个二进制值。

Binary.Combine(binaries as list) as binary

● Binary.Compress

使用给定的压缩类型压缩二进制值。此调用的结果是输入的压缩后副本。压缩类型包括:

Compression.GZip
Compression.Deflate

Binary.Compress(binary as nullable binary, compressionType as number) as nullable binary

压缩二进制值。

Binary.Compress(Binary.FromList(List.Repeat({10}, 1000)), Compression.Deflate)
#binary({227, 226, 26, 5, 163, 96, 20, 12, 119, 0, 0})

● Binary.Decompress

使用给定压缩类型解压缩二进制值。此调用的结果是解压缩后的输入副本。压缩类型包括:

Compression.GZip
Compression.Deflate

Binary.Decompress(binary as nullable binary, compressionType as number) as nullable binary

解压缩二进制值。

Binary.Decompress(#binary({115, 103, 200, 7, 194, 20, 134, 36, 134, 74, 134, 84, 6, 0}), Compression.Deflate)
#binary({71, 0, 111, 0, 111, 0, 100, 0, 98, 0, 121, 0, 101, 0})

● Binary.From

从给定的 value 返回 binary 值。如果给定的 value 是 null ， Binary.From 将返回 null 。 如果给定的 value 是 binary ，则返回 value 。以下类型的值可以转换为 binary 值:

text : 文本表示形式的 binary 值。有关详细信息，请参阅 Binary.FromText 。

如果 value 属于任何其他类型，则返回错误。

Binary.From(value as any, optional encoding as nullable number) as nullable binary

获取 "1011" 的 binary 值。

```
Binary.From("1011")
Binary.FromText("1011", BinaryEncoding.Base64)
```

● Binary.FromList

将一组数值转换为一个二进制值。

Binary.FromList(list as list) as binary

● Binary.FromText

返回将文本值 text 转换为二进制值(number 的列表)的结果。可以指定 encoding 以便指示在文本值中使用的编码。

以下 BinaryEncoding 值可用于 encoding 。

BinaryEncoding.Base64 : Base 64 编码

BinaryEncoding.Hex : 十六进制编码

Binary.FromText(text as nullable text, optional encoding as nullable number) as nullable binary

将 "1011" 解码为二进制值。

```
Binary.FromText("1011")
Binary.FromText("1011", BinaryEncoding.Base64)
----------
```

将 "1011" 解码为具有十六进制编码的二进制值。

```
Binary.FromText("1011", BinaryEncoding.Hex)
Binary.FromText("EBE=", BinaryEncoding.Base64)
```

● Binary.Length

返回字符数。

Binary.Length(binary as nullable binary) as nullable number

● Binary.ToList

将一个二进制值转换为一组数值。

Binary.ToList(binary as binary) as list

● Binary.ToText

返回将数值 binary 的二进制列表转换为文本值的结果。或者，可以指定 encoding 以便指示要在生成的文本值中使用的编码

以下 BinaryEncoding 值可用于 encoding 。

BinaryEncoding.Base64 : Base 64 编码

BinaryEncoding.Hex : 十六进制编码

Binary.ToText(binary as nullable binary, optional encoding as nullable number) as nullable text

● Binary.Type

表示所有二进制值的类型。

● BinaryEncoding.Base64

在要求 base-64 编码时要用作编码类型的常量。

● BinaryEncoding.Hex

在要求十六进制编码时要用作编码类型的常量。

● BinaryEncoding.Type

指定二进制编码的类型。

● BinaryFormat.7BitEncodedSignedInteger

读取使用 7 位可变长度编码进行编码的 64 位带符号整数的二进制格式。

BinaryFormat.7BitEncodedSignedInteger(binary as binary) as any

● BinaryFormat.7BitEncodedUnsignedInteger

读取使用 7 位可变长度编码进行编码的 64 位无符号整数的二进制格式。

```
BinaryFormat.7BitEncodedUnsignedInteger( binary  as binary) as any
```

● BinaryFormat.Binary

返回读取二进制值的二进制格式。 如果指定 length ，则该二进制值将包含这些字节。 如果未指定 length ，则该二进制值将包含其余字节。 length 可以指定为数值，也可以指定为置于二进制数据之前的长度的二进制格式。

```
BinaryFormat.Binary(optional  length  as any) as function
```

● BinaryFormat.Byte

读取 8 位无符号整数的二进制格式。

```
BinaryFormat.Byte( binary  as binary) as any
```

● BinaryFormat.ByteOrder

返回具有 binaryFormat 指定的字节顺序的二进制格式。 默认字节顺序是 ByteOrder.BigEndian 。

```
BinaryFormat.ByteOrder( binaryFormat  as function,  byteOrder  as number) as function
```

● BinaryFormat.Choice

返回一个二进制格式，它基于已读取的值选择下一个二进制格式。 由此函数生成的二进制格式值的工作方式分为以下几个阶段:

使用 binaryFormat 参数指定的二进制格式读取一个值。

将该值传递到由 chooseFunction 参数指定的选择函数。

该选择函数检查值并且返回第二个二进制格式。

使用该第二个二进制格式读取第二个值。

如果指定了合并函数，则第一个值和第二个值将传递到该合并函数，然后返回最终生成的值。

如果未指定该合并函数，则返回第二个值。

返回第二个值。

可选的 type 参数指示选择函数将返回的二进制格式的类型。 或者可以指定 type any 、type list 或 type binary 。 如果未指定 type 参数，则使用 type any 。 如果使用了 type list 或 type binary ，则系统可能能够返回流式 binary 或 list 值，而不是缓冲后的值，这可以减少读取该格式所需的内存量。

BinaryFormat.Choice(binaryFormat as function, chooseFunction as function, optional type as nullable type, optional combineFunction as nullable function) as function

读取字节的列表，其中的元素数目由第一个字节确定。

```
let
binaryData = #binary({2, 3, 4, 5}),
listFormat = BinaryFormat.Choice(
BinaryFormat.Byte,
(length) => BinaryFormat.List(BinaryFormat.Byte, length))
in
listFormat(binaryData)
{3, 4}
```

读取字节的列表，其中的元素数目由第一个字节确定，并且保留读取的第一个字节。

```
let
binaryData = #binary({2, 3, 4, 5}),
listFormat = BinaryFormat.Choice(
BinaryFormat.Byte,
(length) => BinaryFormat.Record([
length = length,
list = BinaryFormat.List(BinaryFormat.Byte, length)
]))
in
listFormat(binaryData)
[ length = 2, list = {3, 4} ]
```

读取字节的列表，其中的元素数目通过使用流式列表由第一个字节确定。

```
let
binaryData = #binary({2, 3, 4, 5}),
listFormat = BinaryFormat.Choice(
BinaryFormat.Byte,
(length) => BinaryFormat.List(BinaryFormat.Byte, length),
```

```
type list)
in
listFormat(binaryData)
{3, 4}
```

- **BinaryFormat.Decimal**

读取 .NET 16 字节十进制值的二进制格式。

```
BinaryFormat.Decimal( binary  as binary) as any
```

- **BinaryFormat.Double**

读取 8 字节 IEEE 双精度浮点值的二进制格式。

```
BinaryFormat.Double( binary  as binary) as any
```

- **BinaryFormat.Group**

参数如下所示:

binaryFormat 参数指定键值的二进制格式。

group 参数提供与已知项的组有关的信息。

可选的 extra 参数可用于指定一个函数，该函数将为跟随在任何意外的键之后的值返回一个二进制格式值。如果未指定 extra 参数，则在存在意外的键值时会引发错误。

group 参数指定项定义的列表。每个项定义都是一个列表，包含 3-5 个值，如下所示:

键值。与项相对应的键的值。该值在项组内必须唯一。

项格式。与项的值相对应的二进制格式。其允许每个项都具有不同的格式。

项出现次数。针对该项应在组中出现多少次的 BinaryOccurrence.Type 值。如果必需的项未提供，将会导致错误。像对待意外键值一样处理必需的或可选的重复项。

默认项值(可选)。如果默认项值出现在项定义列表中并且不为 Null，将使用默认项值而非默认值。重复或可选项的默认值为 Null，而重复值的默认值为空列表 { }。

项值转换(可选)。如果在项定义列表中提供了项值转换函数并且不为 Null，将调用该函数以便转换该项值，然后才能返回它。只有在该项出现在输入中的情况下才调用该转换函数(将永远不

会用默认值调用它)。

BinaryFormat.Group(binaryFormat as function, group as list, optional extra as nullable function, optional lastKey as any) as function

下面假定一个键值，它是单字节的，在组中应该有 4 个项，并且所有这些项都在该键后有一个数据字节。 这些项按如下所示出现在输入中:

键 1 是必需的，并且与值 11 一起出现。

键 2 重复，与值 22 一起出现两次，并且导致值{ 22, 22 }。

键 3 是可选的，它不出现，并且导致 Null 值。

键 4 重复，但不出现，并且导致值 { }。

键 5 不是该组的一部分，但与值 55 一起出现一次。 使用键值 5 调用该附加函数，并且返回与该值相对应的格式(BinaryFormat.Byte)。 值 55 被读取和放弃。

```
let
b = #binary(
{
1, 11,
2, 22,
2, 22,
5, 55,
1, 11
}),
f = BinaryFormat.Group(
BinaryFormat.Byte,
{
{ 1, BinaryFormat.Byte, BinaryOccurrence.Required },
{ 2, BinaryFormat.Byte, BinaryOccurrence.Repeating },
{ 3, BinaryFormat.Byte, BinaryOccurrence.Optional },
{ 4, BinaryFormat.Byte, BinaryOccurrence.Repeating }
},
(extra) => BinaryFormat.Byte)
in
f(b)
{ 11, { 22, 22 }, null, { } }
-----------------
```

下面的示例说明项值转换和默认项值。 具有键 1 的重复项使用 List.Sum 对读取的值列表进行求和。 具有键 2 的可选项具有默认值 123，而非 Null。

```
let
b = #binary(
{
...
```

● BinaryFormat.Length

返回一个二进制格式，它列出可读取的数据量。 BinaryFormat.List 和 BinaryFormat.Binary 都可用于读取，直至读取到数据末尾。 BinaryFormat.Length 可用于限制读取的字节数。 binaryFormat 参数指定要限制的二进制格式。 length 参数指定要读取的字节数。 length 参数可以是数值，也可以是指定在要读取的值之前出现的长度值格式的二进制格式值。

```
BinaryFormat.Length( binaryFormat  as function,  length  as any) as function
```

在读取字节列表时，将读取字节数限制为 2。

```
let
binaryData = #binary({1, 2, 3}),
listFormat = BinaryFormat.Length(
BinaryFormat.List(BinaryFormat.Byte), 2)
in
listFormat(binaryData)
{1, 2}
----------------
```

在字节列表时将读取字节数限制为该列表前面的字节值。

```
let
binaryData = #binary({1, 2, 3}),
listFormat = BinaryFormat.Length(
BinaryFormat.List(BinaryFormat.Byte), BinaryFormat.Byte)
in
listFormat(binaryData)
{2}
```

● BinaryFormat.List

返回读取项序列的二进制格式并且返回 list 。 参数 binaryFormat 指定每一项的二进制格式。 有三种方式可用于确定读取的项数: 如果未指定 countOrCondition ，将读取二进制格式，直到没有任何项。 如果 countOrCondition 为数值，则二进制格式将读取此数量的项。 如果 countOrCondition 为函数，将为每个项读取都调用该函数。 该函数返回 true 则继续，返回 false 则停止读取项。 最后的项包括在该列表中。 如果 countOrCondition 为二进制格式，则项计数应处于该列表前，并且使用指定的格式读取计数。

BinaryFormat.List(binaryFormat as function, optional countOrCondition as any) as function

读取字节，直到到达数据末尾。

```
let
binaryData = #binary({1, 2, 3}),
listFormat = BinaryFormat.List(BinaryFormat.Byte)
in
listFormat(binaryData)
{1, 2, 3}
```

读取两个字节。

```
let
binaryData = #binary({1, 2, 3}),
listFormat = BinaryFormat.List(BinaryFormat.Byte, 2)
in
listFormat(binaryData)
{1, 2}
```

读取字节，直到字节值大于或等于 2。

```
let
binaryData = #binary({1, 2, 3}),
listFormat = BinaryFormat.List(BinaryFormat.Byte, (x) => x < 2)
in
listFormat(binaryData)
{1, 2}
```

● BinaryFormat.Null

读取零字节并且返回 null 的二进制格式。

BinaryFormat.Null(binary as binary) as any

● BinaryFormat.Record

返回读取记录的二进制格式。 record 参数指定该记录的格式。 该记录中的各字段可具有不同的二进制格式。 如果某个字段包含的值不是二进制格式值，则不会为该字段读取任何数据，并且该字段值回显到结果。

BinaryFormat.Record(record as record) as function

读取一个记录，该记录包含一个 16 位整数和一个 32 位整数。

```
let
binaryData = #binary({
0x00, 0x01,
0x00, 0x00, 0x00, 0x02}),
recordFormat = BinaryFormat.Record([
A = BinaryFormat.UnsignedInteger16,
B = BinaryFormat.UnsignedInteger32
])
in
recordFormat(binaryData)
[A = 1, B = 2]
```

● BinaryFormat.SignedInteger16

读取 16 位带符号整数的二进制格式。

BinaryFormat.SignedInteger16(binary as binary) as any

● BinaryFormat.SignedInteger32

读取 32 位带符号整数的二进制格式。

BinaryFormat.SignedInteger32(binary as binary) as any

● BinaryFormat.SignedInteger64

读取 64 位带符号整数的二进制格式。

BinaryFormat.SignedInteger64(binary as binary) as any

● BinaryFormat.Single

读取 4 字节 IEEE 单精度浮点值的二进制格式。

BinaryFormat.Single(binary as binary) as any

● BinaryFormat.Text

返回读取文本值的二进制格式。 length 指定要解码的字节数，或者指定要置于文本前面的长度的二进制格式。 可选的 encoding 值指定文本的编码。 如果未指定 encoding ，则根据 Unicode 字节顺序标记确定该编码。 如果不存在字节顺序标记，则使用 TextEncoding.Utf8 。

```
BinaryFormat.Text( length as any, optional encoding as nullable number) as function

    将两个字节解码为 ASCII 文本。

    let
    binaryData = #binary({65, 66, 67}),
    textFormat = BinaryFormat.Text(2, TextEncoding.Ascii)
    in
    textFormat(binaryData)
    "AB"
    ----------------

    对 ASCII 文本进行解码，其中，以字节为单位的文本长度作为一个字节出现在文本之前。

    let
    binaryData = #binary({2, 65, 66}),
    textFormat = BinaryFormat.Text(BinaryFormat.Byte,
    TextEncoding.Ascii)
    in
    textFormat(binaryData)
    "AB"
```

● BinaryFormat.Transform

返回一个二进制格式，该二进制格式将转换由另一个二进制格式读取的值。 参数 binaryFormat 指定将用于读取值的二进制格式。 使用读取的值调用 function ，并返回转换后的值。

```
BinaryFormat.Transform( binaryFormat as function, function as function) as function

    读取一个字节并向其加 1。

    let
    binaryData = #binary({1}),
    transformFormat = BinaryFormat.Transform(
    BinaryFormat.Byte,
    (x) => x + 1)
    in
```

transformFormat(binaryData)
2

- **BinaryFormat.UnsignedInteger16**

读取 16 位无符号整数的二进制格式。

BinaryFormat.UnsignedInteger16(binary as binary) as any

- **BinaryFormat.UnsignedInteger32**

读取 32 位无符号整数的二进制格式。

BinaryFormat.UnsignedInteger32(binary as binary) as any

- **BinaryFormat.UnsignedInteger64**

读取 64 位无符号整数的二进制格式。

BinaryFormat.UnsignedInteger64(binary as binary) as any

- **BinaryOccurrence.Optional**

要求该项在输入中不出现或出现一次。

- **BinaryOccurrence.Repeating**

要求该项在输入中不出现或出现多次。

- **BinaryOccurrence.Required**

要求该项在输入中出现一次。

- **BinaryOccurrence.Type**

指定项应在组中出现的次数。

Optional : 要求该项在输入中不出现或出现一次。

Repeating : 要求该项在输入中不出现或出现多次。

Required : 要求该项在输入中出现一次。

Byte.From

从给定的 value 返回 8 位整数 number 值。如果给定的 value 为 null ， Byte.From 将返回 null 。如果给定的 value 为 8 位整数范围内的 number ，且没有小数部分，将返回 value 。如果有小数部分，则该数字将以指定舍入模式舍入。默认舍入模式为 RoundingMode.ToEven 。如果给定的 value 为其他类型，请参阅 Number.FromText ，将其转换为 number 值，此后有关将 number 值转换为 8 位整数 number 值的上一语句适用。有关可用的舍入模式，请参阅 Number.Round 。

Byte.From(value as any, optional culture as nullable text, optional roundingMode as nullable number) as nullable number

获取 "4" 的 8 位整数 number 值。

```
Byte.From("4")
4
----------
```

使用 RoundingMode.AwayFromZero 获取 "4.5" 的 8 位整数 number 值。

```
Byte.From("4.5", null, RoundingMode.AwayFromZero)
5
```

- ## Byte.Type

表示所有字节的类型。

- ## ByteOrder.BigEndian

BinaryFormat.ByteOrder 中 byteOrder 参数的可能值。 最重要字节在 Big Endian 字节顺序中最先出现。

- ## ByteOrder.LittleEndian

BinaryFormat.ByteOrder 中 byteOrder 参数的可能值。 最不重要字节在 Little Endian 字节顺序中最先出现。

- ## ByteOrder.Type

指定字节顺序。

Little Endian：最不重要的字节最先出现。

Big Endian：最重要的字节最先出现。

Character.FromNumber

返回与该数值等效的字符。

Character.FromNumber(number as nullable number) as nullable text

假定提供数值 9，则查找字符值。

```
Character.FromNumber(9)
"#(tab)"
```

● Character.ToNumber

返回与字符 character 等效的数值。

Character.ToNumber(character as nullable text) as nullable number

假定提供字符 "#(tab)" 9，则查找数值。

```
Character.ToNumber("#(tab)")
9
```

● Character.Type

表示所有字符的类型。

● Combiner.CombineTextByDelimiter

返回一个函数，它使用指定的分隔符将文本列表合并成单个文本。

Combiner.CombineTextByDelimiter(delimiter as text, optional quoteStyle as nullable number) as function

● Combiner.CombineTextByEachDelimiter

返回一个函数，它按顺序使用每个指定的分隔符将文本列表合并成单个文本。

Combiner.CombineTextByEachDelimiter(delimiters as list, optional quoteStyle as nullable number) as function

● Combiner.CombineTextByLengths

返回一个函数，它使用指定的长度将文本列表合并成单个文本。

Combiner.CombineTextByLengths(lengths as list, optional template as nullable text) as function

- **Combiner.CombineTextByPositions**

返回一个函数，它使用指定的位置将文本列表合并成单个文本。

Combiner.CombineTextByPositions(positions as list, optional template as nullable text) as function

- **Combiner.CombineTextByRanges**

返回一个函数，它使用指定的位置和长度将文本列表合并成单个文本。

Combiner.CombineTextByRanges(ranges as list, optional template as nullable text) as function

- **Comparer.Equals**

使用提供的 comparer 对两个给定值(x 和 y)进行同等性检查，从而返回一个 logical 值。

comparer 是用于控制比较的 Comparer 。比较器可用于提供不区分大小写的比较或识别区域性与区域设置的比较。

公式语言中提供了以下内置比较器:

Comparer.Ordinal : 用于执行完全序号比较

Comparer.OrdinalIgnoreCase : 用于执行不区分大小写的完全序号比较

Comparer.FromCulture : 用于执行识别区域的比较

Comparer.Equals(comparer as function, x as any, y as any) as logical

使用 "en-US" 区域设置比较 "1" 和 "A" 以确定这些值是否相等。

```
Comparer.Equals(Comparer.FromCulture("en-us"), "1", "A")
false
```

- **Comparer.FromCulture**

返回一个比较器函数，得出用于比较的区分大小写的 culture 和逻辑值 ignoreCase 。

ignoreCase 的默认值为 false。区域性的值是 .NET 框架所使用的的区域设置的已知文本表示。

Comparer.FromCulture(culture as text, optional ignoreCase as nullable logical) as function

使用 "en-US" 区域设置比较 "a" 和 "A" 以确定这些值是否相等。

```
Comparer.FromCulture("en-us")("a", "A")
-1
```

使用忽略大小写的 "en-US" 区域设置比较 "a" 和 "A" 以确定这些值是否相等。

```
Comparer.FromCulture("en-us", true)("a", "A")
0
```

● Comparer.Ordinal

返回一个使用 Ordinal 规则的比较器函数以比较提供的值 x 和 y 。

Comparer.Ordinal(x as any, y as any) as number

使用 Ordinal 规则，比较 "encyclopædia" 和 "encyclopaedia" 是否相等。请注意，这些与使用 Comparer.FromCulture("en-us") 等效。

```
Comparer.Equals(Comparer.Ordinal, "encyclopædia", "encyclopaedia")
false
```

● Comparer.OrdinalIgnoreCase

返回使用 Ordinal 规则来比较提供的值 x 和 y 的不区分大小写的比较器函数。

Comparer.OrdinalIgnoreCase(x as any, y as any) as number

使用不区分大小写的 Ordinal 规则，比较 "Abc" 和 "abc"。请注意，通过使用 Comparer.Ordinal，"Abc" 要小于 "abc"。

```
Comparer.OrdinalIgnoreCase("Abc", "abc")
0
```

● Compression.Deflate

压缩数据为 "Deflate" 格式。

● Compression.GZip

压缩数据为 "GZip" 格式。

- **Compression.Type**

指定压缩类型。

- **Csv.Document**

返回表形式的 CSV 文档内容。

columns :如果指定了一个记录，且 delimiter 、 extraValues 、 encoding 为 Null，则所有参数 Delimiter 、 Columns 、 Encoding 、 CsvStyle 和 QuoteStyle 都将从该记录中获取。

delimiter 可以采用单个字符或列表;如果未指定或为 Null，则使用逗号。

有关可选 extraValues 的支持值，请参阅 ExtraValues.Type 。

encoding 指定文本编码类型。

```
Csv.Document( source as any, optional columns as any, optional delimiter as any, optional extraValues as nullable number, optional encoding as nullable number) as table
```

处理包含列标题的 CSV 文本

```
Table.PromoteHeaders(Csv.Document("OrderID,Item
1,Fishing rod
2,1 lb. worms"))
Table.FromRecords({
[ OrderID = "1", Item = "Fishing rod" ],
[ OrderID = "2", Item = "1 lb. worms" ]
})
```

- **CsvStyle.QuoteAfterDelimiter**

字段中的引用只有在紧跟分隔符时才有意义。

- **CsvStyle.QuoteAlways**

不论在何处出现，字段中的引用始终有意义。

- **CsvStyle.Type**

说明 Csv 文档中引号的重要性。

QuoteAfterDelimiter：字段中的引号仅在紧跟分隔符之后时有意义。

QuoteAlways：字段中的引号出现在任何位置时均有意义。

● Cube.AddAndExpandDimensionColumn

将指定维度表 dimensionSelector 合并到多维数据集 cube 的筛选上下文中，并通过展开指定维度属性集 attributeNames 来更改维度粒度。维度属性将添加到含有名为 newColumnNames （如果未指定，则为 attributeNames ）的列的表格视图。

Cube.AddAndExpandDimensionColumn(cube as table, dimensionSelector as any, attributeNames as list, optional newColumnNames as any) as table

● Cube.AddMeasureColumn

将名为 column 的列添加到 cube ，其中包含在每行的行上下文中应用的度量值 measureSelector 的结果。度量值应用受维度粒度和切片的变化的影响。执行特定多维数据集操作后，系统会调整度量值。

Cube.AddMeasureColumn(cube as table, column as text, measureSelector as any) as table

● Cube.ApplyParameter

通过 arguments 将 parameter 应用到 cube 后返回多维数据集。

Cube.ApplyParameter(cube as table, parameter as any, optional arguments as nullable list) as table

● Cube.AttributeMemberId

从成员属性值返回唯一的成员标识符。 value 。对其他值返回 null。

Cube.AttributeMemberId(value as any) as any

● Cube.CollapseAndRemoveColumns

通过折叠映射至指定列 columnNames 的属性，更改 cube 的筛选上下文的维度粒度。还将从多维数据集的表格视图中删除列。

Cube.CollapseAndRemoveColumns(cube as table, columnNames as list) as table

● Cube.Dimensions

返回包含 cube 中可用的维度集的表。每个维度均为包含维度属性集的表，而每个维度属性则表现为维度表中的一列。使用 Cube.AddAndExpandDimensionColumn 可展开多维数据集中的维度。

Cube.Dimensions(cube as table) as table

- **Cube.DisplayFolders**

返回可在 cube 中使用的表示对象(如维度和度量值)显示文件夹层次结构的表嵌套树。

Cube.DisplayFolders(cube as table) as table

- **Cube.Measures**

返回包含 cube 中可用的度量值集的表。

每个度量值表示为一个函数。使用 Cube.AddMeasureColumn 可将度量值应用于多维数据集。

Cube.Measures(cube as table) as table

- **Cube.Parameters**

返回的表所包含的参数集可应用到 cube 。每个参数都是一个函数，可调用这些函数以获取 cube ，其中应用了形参及其实参。

Cube.Parameters(cube as table) as table

- **Cube.Transform**

在 cube 上应用多维数据集函数列表 transforms 。

Cube.Transform(cube as table, transforms as list) as table

- **Culture.Current**

返回应用程序的当前区域性的名称。

- **Currency.From**

从给定的 value 返回 currency 值。如果给定的 value 为 null ，则 Currency.From 将返回 null 。如果给定的 value 是货币范围内的 number ，则将 value 的小数部分舍入为 4 位小数位数后返回。如果给定的 value 为任何其他类型，请参阅

Number.FromText 以将其转换为 number 值，随后之前有关将 number 值转换为 currency 值的语句适用。货币的有效范围为 -922,337,203,685,477.5808 到 922,337,203,685,477.5807 。有关可用的舍入模式，请参阅 Number.Round ，默认值为 RoundingMode.ToEven 。

Currency.From(value as any, optional culture as nullable text, optional roundingMode as nullable number) as nullable number

获取 "1.23455" 的 currency 值。

```
Currency.From("1.23455")
1.2346
----------------
```

使用 RoundingMode.Down 获取 "1.23455" 的 currency 值。

```
Currency.From("1.23455", "en-Us", RoundingMode.Down)
1.2345
```

● Currency.Type

表示货币值的类型。

● DB2.Database

返回 DB2 数据库中(在名为 database 的数据库实例的服务器 server 上)可用的 SQL 表和视图的表。可以视情况使用服务器指定端口，并用冒号分隔。可以指定可选记录参数 options 来控制以下选项:

CreateNavigationProperties : 一个逻辑值(true/false)，用于设置是否在返回的值上生成导航属性(默认值为 true)。

NavigationPropertyNameGenerator : 一个函数，用于创建导航属性的名称。

Query : 被转换为在服务器上运行的 SQL 查询的文本。可以指定多个查询，但是将只返回第一个结果。

CommandTimeout : 一个时间段，控制在取消服务器端查询之前允许该查询运行的时间。默认值为十分钟。

HierarchicalNavigation : 一个逻辑值(true/false)，用于设置是否查看按架构名称分组的表(默认值为 false)。

Implementation : 指定要使用的内部数据库提供程序实现。有效值为: "IBM" 和 "Microsoft"。

BinaryCodePage : CCSID (编码字符集标识符)的一个值，可将 DB2 FOR BIT 二进制数据解码为字符串。适用于 Implementation = “Microsoft"。设置为 0 将禁用转换(默认值)。设置为 1 将基于数据库编码进行转换。设置其他 CCSID 编号将转换为应用程序编码。

PackageCollection : 为包集合指定字符串值(默认值为"NULLID")以启用处理 SQL 语句所需的共享包。适用于 Implementation = "Microsoft"。

...

DB2.Database(server as text, database as text, optional options as nullable record) as table

● Date.AddDays

通过将 numberOfDays 天添加到 datetime 值 dateTime ，返回 date 、 datetime 或 datetimezone 结果。

dateTime : 天数要添加到的 date 、 datetime 或 datetimezone 值。

numberOfDays : 要添加的天数。

Date.AddDays(dateTime as any, numberOfDays as number) as any

将 5 天添加到表示日期 5/14/2011 的 date、datetime 或 datetimezone 值。

Date.AddDays(#date(2011, 5, 14), 5)
#date(2011, 5, 19)

● Date.AddMonths

通过将 numberOfMonths 个月添加到 datetime 值 dateTime ，返回 date 、 datetime 或 datetimezone 结果。

dateTime : 月份数要添加到的 date 、 datetime 或 datetimezone 值。

numberOfMonths : 要添加的月份数。

Date.AddMonths(dateTime as any, numberOfMonths as number) as any

将 5 个月添加到表示日期 5/14/2011 的 date、datetime 或 datetimezone 值。

Date.AddMonths(#date(2011, 5, 14), 5)
#date(2011, 10, 14)

将 18 个月添加到表示日期和时间 5/14/2011 08:15:22 AM 的 date、datetime 或 datetimezone 值。

Date.AddMonths(#datetime(2011, 5, 14, 8, 15, 22), 18)
#datetime(2012, 11, 14, 8, 15, 22)

● Date.AddQuarters

通过将 numberOfQuarters 个季度添加到 datetime 值 dateTime ，返回 date 、 datetime 或 datetimezone 结果。

dateTime : 季度数要添加到的 date 、 datetime 或 datetimezone 值。

numberOfQuarters : 要添加的季度数。

Date.AddQuarters(dateTime as any, numberOfQuarters as number) as any

将 1 个季度添加到表示日期 5/14/2011 的 date、datetime 或 datetimezone 值。

Date.AddQuarters(#date(2011, 5, 14), 1)
#date(2011, 8, 14)

● Date.AddWeeks

通过将 numberOfWeeks 个星期添加到 datetime 值 dateTime ，返回 date 、 datetime 或 datetimezone 结果。

dateTime : 星期数要添加到的 date 、 datetime 或 datetimezone 值。

numberOfWeeks : 要添加的星期数。

Date.AddWeeks(dateTime as any, numberOfWeeks as number) as any

将 2 个星期添加到表示日期 5/14/2011 的 date、datetime 或 datetimezone 值。

```
Date.AddWeeks(#date(2011, 5, 14), 2)
#date(2011, 5, 28)
```

● Date.AddYears

返回将 numberOfYears 添加到 datetime 值 dateTime 的 date 、 datetime 或 datetimezone 结果。

dateTime : 年份数要添加到的 date 、 datetime 或 datetimezone 值。

numberOfYears : 要添加的年份数。

```
Date.AddYears( dateTime as any, numberOfYears as number) as any
```

将 4 年添加到表示日期 5/14/2011 的 date、datetime 或 datetimezone 值。

```
Date.AddYears(#date(2011, 5, 14), 4)
#date(2015, 5, 14)
```

将 10 年添加到表示日期和时间 5/14/2011 08:15:22 AM 的 date、datetime 或 datetimezone 值。

```
Date.AddYears(#datetime(2011, 5, 14, 8, 15, 22), 10)
#datetime(2021, 5, 14, 8, 15, 22)
```

● Date.Day

返回 date 、 datetime 或 datetimezone 值的日部分。

dateTime : 要从其提取日部分的 date 、 datetime 或 datetimezone 值。

```
Date.Day( dateTime as any) as nullable number
```

获取表示 5/14/2011 05:00:00 PM 的日期和时间的 date、datetime 或 datetimezone 值的日部分。

```
Date.Day(#datetime(2011, 5, 14, 17, 0, 0))
14
```

● Date.DayOfWeek

返回介于 0 和 6 之间的数值，该值在提供的日期时间值 dateTime 中表示星期几。

此函数取可选的日期值 firstDayOfWeek ，以便为此相对计算设置一周的第一天。firstDay 的默认值是 Day.Sunday。

有效值为: Day.Sunday、Day.Monday、Day.Tuesday、Day.Wednesday、Day.Thursday、Day.Friday 和 Day.Saturday。

dateTime : 从其确定星期几的 date 、 datetime 或 datetimezone 值。

firstDayOfWeek : 表示此计算的一周的第一天的 Day 类型。

```
Date.DayOfWeek( dateTime  as any, optional  firstDayOfWeek  as nullable number) as any
```

```
获取 2011 年 2 月 21 日是星期几，(默认值) Sunday 是该周的第一天。
Date.DayOfWeek(#date(2011, 02, 21))
1
----------
获取 2011 年 2 月 21 日是星期几，Monday 是该周的第一天。
Date.DayOfWeek(#date(2011, 02, 21), Day.Monday)
0
```

● Date.DayOfWeekName

针对所提供的 date 和可选的区域性 culture 返回星期几。

```
Date.DayOfWeekName( date  as any, optional  culture  as nullable text) as nullable text
```

```
获取星期几。
Date.DayOfWeekName(#date(2011, 12, 31), "en-US")
"Saturday"
```

● Date.DayOfYear

返回一个数值，该值在提供的 date 、 datetime 或 datetimezone 值 dateTime 中表示一年中的日期。

```
Date.DayOfYear( dateTime  as any) as nullable number
```

```
2011 年 3 月 1 日这一天所代表的数值(#date(2011, 03, 01))。
Date.DayOfYear(#date(2011, 03, 01))
60
```

● Date.DaysInMonth

返回 date 、 datetime 或 datetimezone 值 dateTime 中一个月的天数。

dateTime : 为其返回月份中的天数的 date 、 datetime 或 datetimezone 值。

Date.DaysInMonth(dateTime as any) as nullable number

由 #date(2011, 12, 01)> 表示的十二月的天数。

```
Date.DaysInMonth(#date(2011, 12, 01))
31
```

● Date.EndOfDay

返回在 dateTime 中表示一天结束的 date 、 datetime 或 datetimezone 值。保留时区信息。

dateTime : 从中计算一天结束值的 date 、 datetime 或 datetimezone 值。

Date.EndOfDay(dateTime as any) as any

获取 5/14/2011 05:00:00 PM 的一天结束值。

```
Date.EndOfDay(#datetime(2011, 5, 14, 17, 0, 0))
#datetime(2011, 5, 14, 23, 59, 59.9999999)
```

获取 5/17/2011 05:00:00 PM -7:00 的一天结束值。

```
Date.EndOfDay(#datetimezone(2011, 5, 17, 5, 0, 0, -7, 0))
#datetimezone(2011, 5, 17, 23, 59, 59.9999999, -7, 0)
```

● Date.EndOfMonth

返回 dateTime 中月份的最后一天。

dateTime : 从中计算月份结束值的 date 、 datetime 或 datetimezone 值。

Date.EndOfMonth(dateTime as any) as any

获取 5/14/2011 的月份结束值。

```
Date.EndOfMonth(#date(2011, 5, 14))
#date(2011, 5, 31)
```

获取 5/17/2011 05:00:00 PM -7:00 的月份结束值。

```
Date.EndOfMonth(#datetimezone(2011, 5, 17, 5, 0, 0, -7, 0))
#datetimezone(2011, 5, 31, 23, 59, 59.9999999, -7, 0)
```

Date.EndOfQuarter

返回在 dateTime 中表示季度结束值的 date 、 datetime 或 datetimezone 值。保留时区信息。

dateTime : 从中计算季度结束值的 date 、 datetime 或 datetimezone 值。

Date.EndOfQuarter(dateTime as any) as any

查找 2011 年 10 月 10 日上午 8:00 (#datetime(2011, 10, 10, 8, 0, 0)) 的季度结束值。

```
Date.EndOfQuarter(#datetime(2011, 10, 10, 8, 0, 0))
#datetime(2011, 12, 31, 23, 59, 59.9999999)
```

Date.EndOfWeek

返回提供的 date 、 datetime 或 datetimezone dateTime 中星期的最后一天。

此函数取可选的 Day firstDayOfWeek ，以便为此相对计算设置一周的第一天。默认值为 Day.Sunday 。

dateTime : 从中计算星期的最后一天的 date 、 datetime 或 datetimezone 值。

firstDayOfWeek : [可选] 表示星期的第一天的 Day.Type 值。可能值为 Day.Sunday 、 Day.Monday 、 Day.Tuesday 、 Day.Wednesday 、 Day.Thursday 、 Day.Friday 和 Day.Saturday. 。默认值为 Day.Sunday 。

Date.EndOfWeek(dateTime as any, optional firstDayOfWeek as nullable number) as any

获取 5/14/2011 的星期结束值。

```
Date.EndOfWeek(#date(2011, 5, 14))
#date(2011, 5, 14)
----------
```

获取 5/17/2011 05:00:00 PM -7:00 的星期结束值，Sunday 作为该星期的第一天。

```
Date.EndOfWeek(#datetimezone(2011, 5, 17, 5, 0, 0, -7, 0), Day.Sunday)
#datetimezone(2011, 5, 21, 23, 59, 59.9999999, -7, 0)
```

Date.EndOfYear

返回表示 dateTime 中的年份结束值的一个值，包括秒的小数形式。保留时区信息。

dateTime：从中计算年份结束值的 date 、 datetime 或 datetimezone 值。

Date.EndOfYear(dateTime as any) as any

获取 5/14/2011 05:00:00 PM 的年份结束值。

```
Date.EndOfYear(#datetime(2011, 5, 14, 17, 0, 0))
#datetime(2011, 12, 31, 23, 59, 59.9999999)
```

获取 5/17/2011 05:00:00 PM -7:00 的小时结束值。

```
Date.EndOfYear(#datetimezone(2011, 5, 17, 5, 0, 0, -7, 0))
#datetimezone(2011, 12, 31, 23, 59, 59.9999999, -7, 0)
```

● Date.From

从给定的 value 返回 date 值。如果给定的 value 是 null ， Date.From 将返回 null 。 如果给定的 value 是 date ，则返回 value 。以下类型的值可以转换为 date 值:

text：文本表示形式的 date 值。有关详细信息，请参阅 Date.FromText 。

datetime： value 的日期部分。

datetimezone：本地日期时间中等效于 value 的日期部分。

number：等效于 value 表示的 OLE 自动化日期的日期时间的日期部分。

如果 value 属于任何其他类型，则返回错误。

Date.From(value as any, optional culture as nullable text) as nullable date

将 43910 转换为 date 值。

```
Date.From(43910)
#date(2020, 3, 20)
```

将 #datetime(1899, 12, 30, 06, 45, 12) 转换为 date 值。

```
Date.From(#datetime(1899, 12, 30, 06, 45, 12))
#date(1899, 12, 30)
```

● Date.FromText

按照 ISO 8601 格式标准，从文本表示形式 text 创建 date 值。

Date.FromText("2010-02-19") // 日期，yyyy-MM-dd

Date.FromText(text as nullable text, optional culture as nullable text) as nullable date

将 "December 31, 2010" 转换为日期值。

```
Date.FromText("2010-12-31")
#date(2010, 12, 31)
```

将 "December 31, 2010" 转换为不同格式的 date 值

```
Date.FromText("2010, 12, 31")
#date(2010, 12, 31)
```

将 "December, 2010" 转换为 date 值。

```
Date.FromText("2010, 12")
#date(2010, 12, 1)
```

将 "2010" 转换为 date 值。

```
Date.FromText("2010")
#date(2010, 1, 1)
```

● Date.IsInCurrentDay

指示在当天中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInCurrentDay(dateTime as any) as nullable logical

确定当前系统时间是否处于当天。

```
Date.IsInCurrentDay(DateTime.FixedLocalNow())
true
```

● Date.IsInCurrentMonth

指示在当月中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInCurrentMonth(dateTime as any) as nullable logical

确定当前系统时间是否处于当月。

Date.IsInCurrentMonth(DateTime.FixedLocalNow())
true

- **Date.IsInCurrentQuarter**

指示在当前季度中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInCurrentQuarter(dateTime as any) as nullable logical

确定当前系统时间是否处于当前季度。

Date.IsInCurrentQuarter(DateTime.FixedLocalNow())
true

- **Date.IsInCurrentWeek**

指示在当前星期中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInCurrentWeek(dateTime as any) as nullable logical

确定当前系统时间是否处于当前星期。

Date.IsInCurrentWeek(DateTime.FixedLocalNow())
true

- **Date.IsInCurrentYear**

指示在当前年份中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInCurrentYear(dateTime as any) as nullable logical

确定当前系统时间是否处于当前年份。

Date.IsInCurrentYear(DateTime.FixedLocalNow())
true

- **Date.IsInNextDay**

指示在下一天中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInNextDay(dateTime as any) as nullable logical

确定当前系统时间之后的那一天是否处于下一天。

Date.IsInNextDay(Date.AddDays(DateTime.FixedLocalNow(), 1))
true

● Date.IsInNextMonth

指示在下一月中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInNextMonth(dateTime as any) as nullable logical

确定当前系统时间之后的那一月是否处于下一月。

Date.IsInNextMonth(Date.AddMonths(DateTime.FixedLocalNow(), 1))
true

● Date.IsInNextNDays

表示根据系统当前日期和时间确定，给定的日期时间值 dateTime 是否会出现在接下来的几天中。

dateTime：要评估的 date 、 datetime 或 datetimezone 值。

days：天数。

Date.IsInNextNDays(dateTime as any, days as number) as nullable logical

确定当前系统时间之后的日期是否在接下来的两天中。

Date.IsInNextNDays(Date.AddDays(DateTime.FixedLocalNow(), 1), 2)
true

● Date.IsInNextNMonths

表示根据系统当前日期和时间确定，给定的日期时间值 dateTime 是否会出现在接下来的几个月中。

dateTime：要评估的 date 、 datetime 或 datetimezone 值。

months：月数。

Date.IsInNextNMonths(dateTime as any, months as number) as nullable logical

确定当前系统时间之后的月份是否在接下来的两个月中。

Date.IsInNextNMonths(Date.AddMonths(DateTime.FixedLocalNow(), 1), 2)
true

● Date.IsInNextNQuarters

表示根据系统当前日期和时间确定，给定的日期时间值 dateTime 是否会出现在接下来的几个季度中。

dateTime：要评估的 date 、 datetime 或 datetimezone 值。

quarters：季度数。

Date.IsInNextNQuarters(dateTime as any, quarters as number) as nullable logical

确定当前系统时间之后的季度是否在接下来的两个季度中。

Date.IsInNextNQuarters(Date.AddQuarters(DateTime.FixedLocalNow(), 1), 2)
true

● Date.IsInNextNWeeks

表示根据系统当前日期和时间确定，给定的日期时间值 dateTime 是否会出现在接下来的几周中。

dateTime：要评估的 date 、 datetime 或 datetimezone 值。

weeks：周数。

Date.IsInNextNWeeks(dateTime as any, weeks as number) as nullable logical

确定当前系统时间之后的周是否在接下来的两周中。

Date.IsInNextNWeeks(Date.AddDays(DateTime.FixedLocalNow(), 7), 2)
true

● Date.IsInNextNYears

表示根据系统当前日期和时间确定，给定的日期时间值 dateTime 是否会出现在接下来的几年中。

dateTime：要评估的 date 、 datetime 或 datetimezone 值。

years：年数。

Date.IsInNextNYears(dateTime as any, years as number) as nullable logical

确定当前系统时间之后的年份是否在接下来的两年中。

```
Date.IsInNextNYears(Date.AddYears(DateTime.FixedLocalNow(), 1), 2)
true
```

● Date.IsInNextQuarter

指示在下一季度中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInNextQuarter(dateTime as any) as nullable logical

确定当前系统时间之后的那一季度是否处于下一季度。

```
Date.IsInNextQuarter(Date.AddQuarters(DateTime.FixedLocalNow(), 1))
true
```

● Date.IsInNextWeek

指示在下一星期中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInNextWeek(dateTime as any) as nullable logical

确定当前系统时间之后的那一星期是否处于下一星期。

```
Date.IsInNextWeek(Date.AddDays(DateTime.FixedLocalNow(), 7))
true
```

● Date.IsInNextYear

指示在下一年度中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInNextYear(dateTime as any) as nullable logical

确定当前系统时间之后的那一年度是否处于下一年度。

```
Date.IsInNextYear(Date.AddYears(DateTime.FixedLocalNow(), 1))
true
```

● Date.IsInPreviousDay

指示在前一天中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInPreviousDay(dateTime as any) as nullable logical

确定当前系统时间之前的那一天是否处于前一天。

```
Date.IsInPreviousDay(Date.AddDays(DateTime.FixedLocalNow(), -1))
true
```

● Date.IsInPreviousMonth

指示在上个月中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInPreviousMonth(dateTime as any) as nullable logical

确定当前系统时间之前的那个月是否处于上个月。

```
Date.IsInPreviousMonth(Date.AddMonths(DateTime.FixedLocalNow(), -1))
true
```

● Date.IsInPreviousNDays

表示根据系统当前日期和时间确定，给定的日期时间值 dateTime 是否会出现在之前的几天中。

dateTime：要评估的 date 、 datetime 或 datetimezone 值。

days：天数。

Date.IsInPreviousNDays(dateTime as any, days as number) as nullable logical

确定当前系统时间之前的日期是否在之前的两天中。

Date.IsInPreviousNDays(Date.AddDays(DateTime.FixedLocalNow(), -1), 2)
true

● Date.IsInPreviousNMonths

表示根据系统当前日期和时间确定，给定的日期时间值 dateTime 是否会出现在之前的几个月中。

dateTime：要评估的 date 、 datetime 或 datetimezone 值。

months：月数。

Date.IsInPreviousNMonths(dateTime as any, months as number) as nullable logical

确定当前系统时间之前的月份是否在之前的两个月中。

Date.IsInPreviousNMonths(Date.AddMonths(DateTime.FixedLocalNow(), -1), 2)
true

● Date.IsInPreviousNQuarters

表示根据系统当前日期和时间确定，给定的日期时间值 dateTime 是否会出现在之前的几个季度中。

dateTime：要评估的 date 、 datetime 或 datetimezone 值。

quarters：季度数。

Date.IsInPreviousNQuarters(dateTime as any, quarters as number) as nullable logical

确定当前系统时间之前的季度是否在之前的两个季度中。

Date.IsInPreviousNQuarters(Date.AddQuarters(DateTime.FixedLocalNow(), -1), 2)
true

● Date.IsInPreviousNWeeks

表示根据系统当前日期和时间确定，给定的日期时间值 dateTime 是否会出现在之前的几周中。

dateTime：要评估的 date 、 datetime 或 datetimezone 值。

weeks：周数。

Date.IsInPreviousNWeeks(dateTime as any, weeks as number) as nullable logical

确定当前系统时间之前的周是否在之前的两周中。

```
Date.IsInPreviousNWeeks(Date.AddDays(DateTime.FixedLocalNow(), -7), 2)
true
```

● Date.IsInPreviousNYears

表示根据系统当前日期和时间确定，给定的日期时间值 dateTime 是否会出现在之前的几年中。

dateTime：要评估的 date 、 datetime 或 datetimezone 值。

years：年数。

Date.IsInPreviousNYears(dateTime as any, years as number) as nullable logical

确定当前系统时间之前的年份是否在之前的两年中。

```
Date.IsInPreviousNYears(Date.AddYears(DateTime.FixedLocalNow(), -1), 2)
true
```

● Date.IsInPreviousQuarter

指示在上个季度中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInPreviousQuarter(dateTime as any) as nullable logical

确定当前系统时间之前的那个季度是否处于上个季度。

```
Date.IsInPreviousQuarter(Date.AddQuarters(DateTime.FixedLocalNow(), -1))
true
```

Date.IsInPreviousWeek

指示在上个星期中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInPreviousWeek(dateTime as any) as nullable logical

确定当前系统时间之前的那个星期是否处于上个星期。

```
Date.IsInPreviousWeek(Date.AddDays(DateTime.FixedLocalNow(), -7))
true
```

Date.IsInPreviousYear

指示在前一年中给定的日期时间值 dateTime 是否出现，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInPreviousYear(dateTime as any) as nullable logical

确定当前系统时间之前的那个年度是否处于前一年。

```
Date.IsInPreviousYear(Date.AddYears(DateTime.FixedLocalNow(), -1))
true
```

Date.IsInYearToDate

指示在当前年份中给定日期时间值 dateTime 是否出现以及该日期时间值是否就在当天或早于当天，它由系统上的当前日期和时间确定。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

Date.IsInYearToDate(dateTime as any) as nullable logical

确定当前系统时间是否处于本年度截至现在的日期内。

```
Date.IsInYearToDate(DateTime.FixedLocalNow())
true
```

Date.IsLeapYear

指示给定的日期时间值 dateTime 是否处于闰年。

dateTime：要进行求值的 date 、 datetime 或 datetimezone 值。

```
Date.IsLeapYear( dateTime  as any) as nullable logical

    确定由 #date(2012, 01, 01) 表示的 2012 年是否为闰年。

    Date.IsLeapYear(#date(2012, 01, 01))
    true
```

- **Date.Month**

返回提供的 datetime 值 dateTime 的月份部分。

```
Date.Month( dateTime  as any) as nullable number

    查找 #datetime(2011, 12, 31, 9, 15, 36) 中的月份。

    Date.Month(#datetime(2011, 12, 31, 9, 15, 36))
    12
```

- **Date.MonthName**

针对所提供的 date 和可选的区域性 culture 返回月份名称部分。

```
Date.MonthName( date  as any, optional  culture  as nullable text) as nullable text

    获取月份名称。

    Date.MonthName(#datetime(2011, 12, 31, 5, 0, 0), "en-US")
    "December"
```

- **Date.QuarterOfYear**

返回一个介于 1 到 4 之间的数值，该数值指示日期 dateTime 属于年份中的哪个季度。

dateTime 可以是 date 、 datetime 或 datetimezone 值。

```
Date.QuarterOfYear( dateTime  as any) as nullable number

    查找日期 #date(2011, 12, 31) 属于年份中的哪个季度。

    Date.QuarterOfYear(#date(2011, 12, 31))
    4
```

- **Date.StartOfDay**

返回日期 dateTime 的第一个值。

dateTime 必须是 date 、 datetime 或 datetimezone 值。

Date.StartOfDay(dateTime as any) as any

查找 2011 年 10 月 10 日上午 8:00 (#datetime(2011, 10, 10, 8, 0, 0))的日期开始值。

Date.StartOfDay(#datetime(2011, 10, 10, 8, 0, 0))
#datetime(2011, 10, 10, 0, 0, 0)

● Date.StartOfMonth

返回给定 date 或 datetime 类型的月份的第一个值。

Date.StartOfMonth(dateTime as any) as any

查找 2011 年 10 月 10 日上午 8:10:32 (#datetime(2011, 10, 10, 8, 10, 32))的月份开始值。

Date.StartOfMonth(#datetime(2011, 10, 10, 8, 10, 32))
#datetime(2011, 10, 1, 0, 0, 0)

● Date.StartOfQuarter

返回季度 dateTime 的第一个值。

dateTime 必须是 date 、 datetime 或 datetimezone 值。

Date.StartOfQuarter(dateTime as any) as any

查找 2011 年 10 月 10 日上午 8:00 (#datetime(2011, 10, 10, 8, 0, 0))的季度开始值。

Date.StartOfQuarter(#datetime(2011, 10, 10, 8, 0, 0))
#datetime(2011, 10, 1, 0, 0, 0)

● Date.StartOfWeek

返回给定 date 、 datetime 或 datetimezone 值的星期的第一个值。

Date.StartOfWeek(dateTime as any, optional firstDayOfWeek as nullable number) as any

查找 2011 年 10 月 10 日上午 8:10:32 (#datetime(2011, 10, 10, 8, 10, 32))的星期开始值。

Date.StartOfWeek(#datetime(2011, 10, 10, 8, 10, 32))
#datetime(2011, 10, 9, 0, 0, 0)

● Date.StartOfYear

返回给定 date 、 datetime 或 datetimezone 值的年份的第一个值。

Date.StartOfYear(dateTime as any) as any

查找 2011 年 10 月 10 日上午 8:10:32 (#datetime(2011, 10, 10, 8, 10, 32))的年份开始值。

Date.StartOfYear(#datetime(2011, 10, 10, 8, 10, 32))
#datetime(2011, 1, 1, 0, 0, 0)

Date.ToRecord

返回包含给定日期值 date 的各个部分的记录。

date：要从中计算其各个部分的记录的 date 值。

```
Date.ToRecord( date as date) as record
```

将 #date(2011, 12, 31) 值转换为包含日期值的各个部分的记录。

```
Date.ToRecord(#date(2011, 12, 31))
[
Year = 2011,
Month = 12,
Day = 31
]
```

Date.ToText

返回 date (即日期值 date)的文本表示形式。

此函数采用一个可选格式参数 format 。有关支持的格式的完整列表，请参阅库规范文档。

```
Date.ToText( date as nullable date, optional format as nullable text, optional culture as nullable text) as nullable text
```

获取 #date(2010, 12, 31) 的文本表示形式。

```
Date.ToText(#date(2010, 12, 31))
"12/31/2010"
----------
```

使用格式选项获取 #date(2010, 12, 31) 的文本表示形式。

```
Date.ToText(#date(2010, 12, 31), "yyyy/MM/dd")
"2010/12/31"
```

Date.Type

表示所有日期值的类型。

Date.WeekOfMonth

返回一个介于 1 到 5 之间的数值，该数值指示日期 dateTime 属于月份中的哪一周。

dateTime：为其确定属于月份中的第几周的 datetime 值。

```
Date.WeekOfMonth( dateTime as any, optional firstDayOfWeek as nullable number) as nullable number
```

确定 2011 年 3 月 15 日(#date(2011, 03, 15))属于当月的哪一周。

```
Date.WeekOfMonth(#date(2011, 03, 15))
3
```

- ## Date.WeekOfYear

返回一个介于 1 到 54 之间的数值，该数值指示日期 dateTime 属于年份中的哪一周。

dateTime：为其确定属于年份中的第几周的 datetime 值。

```
Date.WeekOfYear( dateTime as any, optional firstDayOfWeek as nullable number) as nullable number
```

确定 2011 年 3 月 23 日(#date(2011, 03, 23))属于该年的哪一周。

```
Date.WeekOfYear(#date(2011, 03, 23))
13
```

- ## Date.Year

返回提供的 datetime 值 dateTime 的年份部分。

```
Date.Year( dateTime as any) as nullable number
```

查找 #datetime(2011, 12, 31, 9, 15, 36) 中的年份。

```
Date.Year(#datetime(2011, 12, 31, 9, 15, 36))
2011
```

- ## DateTime.AddZone

设置日期时间值 dateTime 的时区信息。时区信息包含 timezoneHours ，也可能会包含 timezoneMinutes 。

```
DateTime.AddZone( dateTime as nullable datetime, timezoneHours as number, optional timezoneMinutes as nullable number) as nullable datetimezone
```

将 #datetime(2010, 12, 31, 11, 56, 02) 的时区信息设置为 7 小时，30 分钟。

```
DateTime.AddZone(#datetime(2010, 12, 31, 11, 56, 02), 7, 30)
#datetimezone(2010, 12, 31, 11, 56, 2, 7, 30)
```

- ## DateTime.Date

返回 dateTime 以及给定 date 、 datetime 或 datetimezone 值的日期部分。

```
DateTime.Date( dateTime as any) as nullable date
```

查找 #datetime(2010, 12, 31, 11, 56, 02) 的日期值。

DateTime.Date(#datetime(2010, 12, 31, 11, 56, 02))
#date(2010, 12, 31)

● DateTime.FixedLocalNow

返回设置为系统上的当前日期和时间的 datetime 值。该值是固定的，因此将不会随着连续调用而更改，这与 DateTime.LocalNow 不同，后者可能会在表达式的执行过程中返回不同值。

DateTime.FixedLocalNow() as datetime

● DateTime.From

从给定的 value 返回 datetime 值。如果给定的 value 是 null ， DateTime.From 将返回 null 。 如果给定的 value 是 datetime ，则返回 value 。以下类型的值可以转换为 datetime 值:

text : 文本表示形式的 datetime 值。有关详细信息，请参阅 DateTime.FromText 。

date : 一个 datetime ，它具有 value 作为日期部分以及 12:00:00 AM 作为时间部分。

datetimezone : 等效于 value 的本地 datetime 。

time : 一个 datetime ，它具有作为日期部分的 0 的 OLE 自动化日期的等效日期以及作为时间部分的 value 。

number : 一个 datetime ，它等效于 value 表示的 OLE 自动化日期。

如果 value 属于任何其他类型，则返回错误。

DateTime.From(value as any, optional culture as nullable text) as nullable datetime

将 #time(06, 45, 12) 转换为 datetime 值。

DateTime.From(#time(06, 45, 12))
#datetime(1899, 12, 30, 06, 45, 12)

将 #date(1975, 4, 4) 转换为 datetime 值。

DateTime.From(#date(1975, 4, 4))
#datetime(1975, 4, 4, 0, 0, 0)

● DateTime.FromFileTime

从 fileTime 值创建一个 datetime 值并且将其转换为本地时区。filetime 是一个 Windows 文件时间值，它表示自公元 1601 年 1 月 1 日(C.E.)通用协调时间(UTC)午夜 12:00 后 经过的 100 纳秒时间间隔的数目。

DateTime.FromFileTime(fileTime as nullable number) as nullable datetime

将 129876402529842245 转换为日期时间值。

```
DateTime.FromFileTime(129876402529842245)
#datetime(2012, 7, 24, 14, 50, 52.9842245)
```

● DateTime.FromText

按照 ISO 8601 格式标准，从文本表示形式 text 创建 datetime 值。

```
DateTime.FromText("2010-12-31T01:30:00")     // yyyy-MM-ddThh:mm:ss
```

DateTime.FromText(text as nullable text, optional culture as nullable text) as nullable datetime

将 "2010-12-31T01:30:25" 转换为 datetime 值。

```
DateTime.FromText("2010-12-31T01:30:25")
#datetime(2010, 12, 31, 1, 30, 25)
```

将 "2010-12-31T01:30" 转换为 datetime 值。

```
DateTime.FromText("2010-12-31T01:30")
#datetime(2010, 12, 31, 1, 30, 0)
```

将 "20101231T013025" 转换为 datetime 值。

```
DateTime.FromText("20101231T013025")
#datetime(2010, 12, 31, 1, 30, 25)
```

将 "20101231T01:30:25" 转换为 datetime 值。

```
DateTime.FromText("20101231T01:30:25")
#datetime(2010, 12, 31, 1, 30, 25)
```

将 "20101231T01:30:25.121212" 转换为 datetime 值。

```
DateTime.FromText("20101231T01:30:25.121212")
#datetime(2010, 12, 31, 1, 30, 25.121212)
```

● DateTime.IsInCurrentHour

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于当前小时内。

dateTime：要计算的 datetime 或 datetimezone 值。

DateTime.IsInCurrentHour(dateTime as any) as nullable logical

确定当前系统时间是否处于当前小时内。

DateTime.IsInCurrentHour(DateTime.FixedLocalNow())
true

● DateTime.IsInCurrentMinute

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于当前分钟内。

dateTime：要计算的 datetime 或 datetimezone 值。

DateTime.IsInCurrentMinute(dateTime as any) as nullable logical

确定当前系统时间是否处于当前分钟内。

DateTime.IsInCurrentMinute(DateTime.FixedLocalNow())
true

● DateTime.IsInCurrentSecond

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于当前秒内。

dateTime：要计算的 datetime 或 datetimezone 值。

DateTime.IsInCurrentSecond(dateTime as any) as nullable logical

确定当前系统时间是否处于当前秒内。

DateTime.IsInCurrentSecond(DateTime.FixedLocalNow())
true

● DateTime.IsInNextHour

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于接下来的一小时内。

dateTime：要计算的 datetime 或 datetimezone 值。

DateTime.IsInNextHour(dateTime as any) as nullable logical

确定当前系统时间后的小时是否处于接下来的一小时内。

DateTime.IsInNextHour(DateTime.FixedLocalNow() + #duration(0,1,0,0))
true

- ## DateTime.IsInNextMinute

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于接下来的一分钟内。

dateTime：要计算的 datetime 或 datetimezone 值。

DateTime.IsInNextMinute(dateTime as any) as nullable logical

确定当前系统时间后的分钟是否处于接下来的一分钟内。

DateTime.IsInNextMinute(DateTime.FixedLocalNow() + #duration(0,0,1,0))
true

- ## DateTime.IsInNextNHours

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于接下来的几小时内。

dateTime：要计算的 datetime 或 datetimezone 值。

hours：小时数。

DateTime.IsInNextNHours(dateTime as any, hours as number) as nullable logical

确定当前系统时间后的小时是否处于接下来的两个小时内。

DateTime.IsInNextNHours(DateTime.FixedLocalNow() + #duration(0,2,0,0), 2)
true

- ## DateTime.IsInNextNMinutes

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于接下来的几分钟内。

dateTime：要计算的 datetime 或 datetimezone 值。

minutes：分钟数。

DateTime.IsInNextNMinutes(dateTime as any, minutes as number) as nullable logical

确定当前系统时间后的分钟是否处于接下来的两分钟内。

DateTime.IsInNextNMinutes(DateTime.FixedLocalNow() + #duration(0,0,2,0), 2)
true

● DateTime.IsInNextNSeconds

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于接下来的几秒内。

dateTime：要计算的 datetime 或 datetimezone 值。

seconds：秒数。

DateTime.IsInNextNSeconds(dateTime as any, seconds as number) as nullable logical

确定当前系统时间后的秒是否处于接下来的两秒内。

DateTime.IsInNextNSeconds(DateTime.FixedLocalNow() + #duration(0,0,0,2), 2)
true

● DateTime.IsInNextSecond

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于接下来的一秒内。

dateTime：要计算的 datetime 或 datetimezone 值。

DateTime.IsInNextSecond(dateTime as any) as nullable logical

确定当前系统时间后的秒是否处于接下来的一秒内。

DateTime.IsInNextSecond(DateTime.FixedLocalNow() + #duration(0,0,0,1))
true

● DateTime.IsInPreviousHour

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于前一小时内。

dateTime：要计算的 datetime 或 datetimezone 值。

DateTime.IsInPreviousHour(dateTime as any) as nullable logical

确定当前系统时间前的小时是否处于前一小时内。

```
DateTime.IsInPreviousHour(DateTime.FixedLocalNow() - #duration(0,1,0,0))
true
```

- **DateTime.IsInPreviousMinute**

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于前一分钟内。

dateTime：要计算的 datetime 或 datetimezone 值。

```
DateTime.IsInPreviousMinute( dateTime as any) as nullable logical
```

确定当前系统时间前的分钟是否处于前一分钟内。

```
DateTime.IsInPreviousMinute(DateTime.FixedLocalNow() - #duration(0,0,1,0))
true
```

- **DateTime.IsInPreviousNHours**

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于之前的几小时内。

dateTime：要计算的 datetime 或 datetimezone 值。

hours：小时数。

```
DateTime.IsInPreviousNHours( dateTime as any, hours as number) as nullable logical
```

确定当前系统时间前的小时是否处于前两个小时内。

```
DateTime.IsInPreviousNHours(DateTime.FixedLocalNow() - #duration(0,2,0,0), 2)
true
```

- **DateTime.IsInPreviousNMinutes**

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于之前的几分钟内。

dateTime：要计算的 datetime 或 datetimezone 值。

minutes：分钟数。

```
DateTime.IsInPreviousNMinutes( dateTime as any, minutes as number) as nullable logical
```

确定当前系统时间前的分钟是否处于前两分钟内。

```
DateTime.IsInPreviousNMinutes(DateTime.FixedLocalNow() - #duration(0,0,2,0), 2)
true
```

- ## DateTime.IsInPreviousNSeconds

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于之前的几秒内。

dateTime：要计算的 datetime 或 datetimezone 值。

seconds：秒数。

```
DateTime.IsInPreviousNSeconds( dateTime as any, seconds as number) as nullable logical
```

确定当前系统时间前的秒是否处于前两秒内。

```
DateTime.IsInPreviousNSeconds(DateTime.FixedLocalNow() - #duration(0,0,0,2), 2)
true
```

- ## DateTime.IsInPreviousSecond

指示给定的日期时间值 dateTime 是否按系统当前日期和时间所确定的那样处于前一秒内。

dateTime：要计算的 datetime 或 datetimezone 值。

```
DateTime.IsInPreviousSecond( dateTime as any) as nullable logical
```

确定当前系统时间前的秒是否处于前一秒内。

```
DateTime.IsInPreviousSecond(DateTime.FixedLocalNow() - #duration(0,0,0,1))
true
```

- ## DateTime.LocalNow

返回设置为系统上的当前日期和时间的 datetime 值。

```
DateTime.LocalNow() as datetime
```

- ## DateTime.Time

返回给定日期时间值 dateTime 的时间部分。

```
DateTime.Time( dateTime as any) as nullable time
```

查找 #datetime(2010, 12, 31, 11, 56, 02) 的时间值。

```
DateTime.Time(#datetime(2010, 12, 31, 11, 56, 02))
#time(11, 56, 2)
```

DateTime.ToRecord

返回包含给定日期时间值 dateTime 的各个部分的记录。

dateTime：要从中计算其各个部分的记录的 datetime 值。

DateTime.ToRecord(dateTime as datetime) as record

将 #datetime(2011, 12, 31, 11, 56, 2) 值转换为包含日期和时间值的记录。

```
DateTime.ToRecord(#datetime(2011, 12, 31, 11, 56, 2))
[
Year = 2011,
Month = 12,
Day = 31,
Hour = 11,
Minute = 56,
Second = 2
]
```

DateTime.ToText

返回 dateTime (即日期时间值 dateTime)的文本表示形式。

此函数采用一个可选格式参数 format 。有关支持的格式的完整列表，请参阅库规范文档。

DateTime.ToText(dateTime as nullable datetime, optional format as nullable text, optional culture as nullable text) as nullable text

获取 #datetime(2011, 12, 31, 11, 56, 2) 的文本表示形式。

```
DateTime.ToText(#datetime(2010, 12, 31, 11, 56, 2))
"12/31/2010 11:56:02 AM"
----------------
```

使用格式选项获取 #datetime(2011, 12, 31, 11, 56, 2)的文本表示形式。

```
DateTime.ToText(#datetime(2010, 12, 31, 11, 56, 2), "yyyy/MM/ddThh:mm:ss")
"2010/12/31T11:56:02"
```

DateTime.Type

表示不含关联时区的所有日期和时间值的类型。

DateTimeZone.FixedLocalNow

返回设置为系统上的当前日期和时间的 datetime 值。返回的值包含表示本地时区的时区信息。该值是固定的，因此将不会随着连续调用而更改，这与 DateTimeZone.LocalNow 不同，后者可能会在表达式的执行过程中返回不同值。

DateTimeZone.FixedLocalNow() as datetimezone

- **DateTimeZone.FixedUtcNow**

返回采用 UTC (GMT 时区)的当前日期和时间。该值是固定的，因此将不会随着连续调用而更改。

DateTimeZone.FixedUtcNow() as datetimezone

- **DateTimeZone.From**

从给定的 value 返回 datetimezone 值。如果给定的 value 是 null ，DateTimeZone.From 将返回 null 。 如果给定的 value 是 datetimezone ，则返回 value 。以下类型的值可以转换为 datetimezone 值:

text : 文本表示形式的 datetimezone 值。有关详细信息，请参阅 DateTimeZone.FromText 。

date : 一个 datetimezone ，它具有作为日期部分的 value 、作为时间部分的 12:00:00 AM 以及与本地时区相对应的偏移量。

datetime : 一个 datetimezone ，它具有作为日期时间的 value 以及与本地时区相对应的偏移量。

time : 一个 datetimezone ，它具有作为日期部分的 0 的 OLE 自动化日期的等效日期、作为时间部分的 value 以及与本地时区相对应的偏移量。

number : 一个 datetimezone ，它具有 value 表示的 OLE 自动化日期的等效日期时间以及与本地时区相对应的偏移量。

如果 value 属于任何其他类型，则返回错误。

DateTimeZone.From(value as any, optional culture as nullable text) as nullable datetimezone

将 "2020-10-30T01:30:00-08:00" 转换为 datetimezone 值。

```
DateTimeZone.From("2020-10-30T01:30:00-08:00")
#datetimezone(2020, 10, 30, 01, 30, 00, -8, 00)
```

● DateTimeZone.FromFileTime

从 fileTime 值创建 datetimezone 值并将其转换为本地时区。filetime 是一个 Windows 文件时间值，它表示自公元 1601 年 1 月 1 日协调世界时(UTC)午夜 12:00 后经过的 100 纳秒时间间隔的数目。

DateTimeZone.FromFileTime(fileTime as nullable number) as nullable datetimezone

将 129876402529842245 转换为 datetimezone 值。

```
DateTimeZone.FromFileTime(129876402529842245)
#datetimezone(2012, 7, 24, 14, 50, 52.9842245, -7, 0)
```

● DateTimeZone.FromText

按照 ISO 8601 格式标准，从文本表示形式 text 创建 datetimezone 值。

```
DateTimeZone.FromText("2010-12-31T01:30:00-08:00")     // yyyy-MM-ddThh:mm:ssZ
```

DateTimeZone.FromText(text as nullable text, optional culture as nullable text) as nullable datetimezone

将 "2010-12-31T01:30:00-08:00" 转换为 datetimezone 值。

```
DateTimeZone.FromText("2010-12-31T01:30:00-08:00")
#datetimezone(2010, 12, 31, 1, 30, 0, -8, 0)
----------
```

将 "2010-12-31T01:30:00.121212-08:00" 转换为 datetimezone 值。

```
DateTimeZone.FromText("2010-12-31T01:30:00.121212-08:00")
#datetimezone(2010, 12, 31, 1, 30, 0.121212, -8, 0)
----------
```

将 "2010-12-31T01:30:00Z" 转换为 datetimezone 值。

```
DateTimeZone.FromText("2010-12-31T01:30:00Z")
#datetimezone(2010, 12, 31, 1, 30, 0, 0, 0)
----------
```

将 "20101231T013000+0800" 转换为 datetimezone 值。

```
DateTimeZone.FromText("20101231T013000+0800")
#datetimezone(2010, 12, 31, 1, 30, 0, 8, 0)
```

DateTimeZone.LocalNow

返回设置为系统上的当前日期和时间的 datetimezone 值。

返回的值包含表示本地时区的时区信息。

```
DateTimeZone.LocalNow() as datetimezone
```

DateTimeZone.RemoveZone

从 dateTimeZone 返回 #datetime 值并且删除时区信息。

```
DateTimeZone.RemoveZone( dateTimeZone as nullable datetimezone) as nullable datetime
```

从值 #datetimezone(2011, 12, 31, 9, 15, 36, -7, 0) 中删除时区信息。

```
DateTimeZone.RemoveZone( #datetimezone(2011, 12, 31, 9, 15, 36,-7, 0))
#datetime(2011, 12, 31, 9, 15, 36)
```

DateTimeZone.SwitchZone

将针对 datetimezone 值 dateTimeZone 的时区信息更改为 timezoneHours 和 timezoneMinutes (可选)提供的新时区信息。

如果 dateTimeZone 没有时区部分，将引发异常。

```
DateTimeZone.SwitchZone( dateTimeZone as nullable datetimezone, timezoneHours as number, optional timezoneMinutes as nullable number) as nullable datetimezone
```

将 #datetimezone(2010, 12, 31, 11, 56, 02, 7, 30) 的时区信息更改为 8 小时。

```
DateTimeZone.SwitchZone(#datetimezone(2010, 12, 31, 11, 56, 02, 7, 30), 8)
#datetimezone(2010, 12, 31, 12, 26, 2, 8, 0)
----------
```

将 #datetimezone(2010, 12, 31, 11, 56, 02, 7, 30) 的时区信息更改为 -30 分钟。

```
DateTimeZone.SwitchZone(#datetimezone(2010, 12, 31, 11, 56, 02, 7, 30), 0, -30)
#datetimezone(2010, 12, 31, 3, 56, 2, 0, -30)
```

DateTimeZone.ToLocal

将 datetimezone 值 dateTimeZone 的时区信息更改为本地时区信息。

如果 dateTimeZone 不具有时区部分，则添加本地时区信息。

```
DateTimeZone.ToLocal( dateTimeZone as nullable datetimezone) as nullable datetimezone
```

将针对 #datetimezone(2010, 12, 31, 11, 56, 02, 7, 30) 的时区信息更改为本地时区(假定 PST)。

DateTimeZone.ToLocal(#datetimezone(2010, 12, 31, 11, 56, 02, 7, 30))
#datetimezone(2010, 12, 31, 12, 26, 2, -8, 0)

● DateTimeZone.ToRecord

返回包含给定 datetimezone 值 dateTimeZone 的各个部分的记录。

dateTimeZone：要从中计算其各个部分的记录的 datetimezone 值。

DateTimeZone.ToRecord(dateTimeZone as datetimezone) as record

将 #datetimezone(2011, 12, 31, 11, 56, 2, 8, 0) 值转换为包含日期、时间和时区值的记录。

```
DateTimeZone.ToRecord(#datetimezone(2011, 12, 31, 11, 56, 2, 8, 0))
[
Year = 2011,
Month = 12,
Day = 31,
Hour = 11,
Minute = 56,
Second = 2,
ZoneHours = 8,
ZoneMinutes = 0
]
```

● DateTimeZone.ToText

返回 dateTimeZone (即 datetimezone 值 dateTimeZone)的文本表示形式。

此函数采用一个可选格式参数 format 。有关支持的格式的完整列表，请参阅库规范文档。

DateTimeZone.ToText(dateTimeZone as nullable datetimezone, optional format as nullable text, optional culture as nullable text) as nullable text

获取 #datetimezone(2011, 12, 31, 11, 56, 2, 8, 0) 的文本表示形式。

```
DateTimeZone.ToText(#datetimezone(2010, 12, 31, 11, 56, 2, 8, 0))
"12/31/2010 11:56:02 AM +08:00"
----------
```

使用格式选项获取 #datetimezone(2010, 12, 31, 11, 56, 2, 10, 12)的文本表示形式。

```
DateTimeZone.ToText(#datetimezone(2010, 12, 31, 11, 56, 2, 10, 12), "yyyy/MM/ddThh:mm:sszzz")
"2010/12/31T11:56:02+10:12"
```

● DateTimeZone.ToUtc

将日期时间值 dateTimeZone 的时区信息更改为 UTC 或通用协调时间时区信息。

如果 dateTimeZone 不具有时区部分，则添加 UTC 时区信息。

DateTimeZone.ToUtc(dateTimeZone as nullable datetimezone) as nullable datetimezone

将 #datetimezone(2010, 12, 31, 11, 56, 02, 7, 30) 的时区信息更改为 UTC 时区。

```
DateTimeZone.ToUtc(#datetimezone(2010, 12, 31, 11, 56, 02, 7, 30))
#datetimezone(2010, 12, 31, 4, 26, 2, 0, 0)
```

- ## DateTimeZone.Type

表示相对于某个时区的所有日期和时间值的类型。

- ## DateTimeZone.UtcNow

返回采用 UTC (GMT 时区)的当前日期和时间。

DateTimeZone.UtcNow() as datetimezone

获取采用 UTC 的当前日期和时间。

```
DateTimeZone.UtcNow()
#datetimezone(2011, 8, 16, 23, 34, 37.745, 0, 0)
```

- ## DateTimeZone.ZoneHours

更改此值的时区。

DateTimeZone.ZoneHours(dateTimeZone as nullable datetimezone) as nullable number

- ## DateTimeZone.ZoneMinutes

更改此值的时区。

DateTimeZone.ZoneMinutes(dateTimeZone as nullable datetimezone) as nullable number

- ## Day.Friday

返回 6，该数值表示星期五。

- ## Day.Monday

返回 2，该数值表示星期一。

- ## Day.Saturday

返回 7，该数值表示星期六。

- ## Day.Sunday

返回 1，该数值表示星期日。

- **Day.Thursday**

返回 5，该数值表示星期四。

- **Day.Tuesday**

返回 3，该数值表示星期二。

- **Day.Type**

指定星期几。

- **Day.Wednesday**

返回 4，该数值表示星期三。

- **Decimal.From**

从给定的 value 返回 Decimal number 值。如果给定的 value 为 null ，Decimal.From 将返回 null 。如果给定的 value 为 Decimal 范围内的 number ，将返回 value ，否则返回错误。如果给定的 value 为其他类型，请参阅 Number.FromText ，将其转换为 number 值，此后有关将 number 值转换为 Decimal number 值的上一语句适用。

```
Decimal.From( value as any, optional culture as nullable text) as nullable number
```

获取 "4.5" 的十进制 number 值。

```
Decimal.From("4.5")
```

4.5

- **Decimal.Type**

表示固定点小数的类型。

- **Diagnostics.ActivityId**

为当前正在运行的计算返回不透明的标识符。

```
Diagnostics.ActivityId() as nullable text
```

Diagnostics.Trace

写入跟踪 message (如果已启用跟踪的话)并返回 value 。可选参数 delayed 指定了是否延迟计算 value 直到跟踪到消息。 traceLevel 可取以下任一值:

TraceLevel.Critical

TraceLevel.Error 、

TraceLevel.Warning 、

TraceLevel.Information 、

TraceLevel.Verbose 。

Diagnostics.Trace(traceLevel as number, message as text, value as any, optional delayed as nullable logical) as any

在调用 Text.From 函数前跟踪消息，并返回结果。

```
Diagnostics.Trace(TraceLevel.Information, "TextValueFromNumber", () => Text.From(123), true)
"123"
```

DirectQueryCapabilities.From

DirectQueryCapabilities.From

DirectQueryCapabilities.From(value as any) as table

Double.From

从给定的 value 返回双精度 number 值。如果给定的 value 为 null ， Double.From 将返回 null 。如果给定的 value 为双精度 范围内的 number ，将返回 value ，否则返回错误。如果给定的 value 为其他类型，请参阅 Number.FromText ，将其转换为 number 值，此后有关将 number 值转换为双精度 number 值的上一语句适用。

Double.From(value as any, optional culture as nullable text) as nullable number

获取 "4" 的 Double number 值。

```
Double.From("4.5")
4.5
```

Double.Type

表示双精度浮点数的类型。

Duration.Days

返回提供的 duration 值 duration 的日期部分。

Duration.Days(duration as nullable duration) as nullable number

查找 #duration(5, 4, 3, 2) 中的日期。

```
Duration.Days(#duration(5, 4, 3, 2))
5
```

Duration.From

从给定的 value 返回 duration 值。如果给定的 value 是 null ， Duration.From 将返回 null 。 如果给定的 value 是 duration ，则返回 value 。以下类型的值可以转换为 duration 值:

text : 来自文本形式的占用时间格式(d.h:m:s)的 duration 值。有关详细信息，请参阅 Duration.FromText 。

number : 与由 value 表示的整数天数和不完整天数等效的 duration 。

如果 value 属于任何其他类型，则返回错误。

Duration.From(value as any) as nullable duration

将 2.525 转换为 duration 值。

```
Duration.From(2.525)
#duration(2, 12, 36, 0)
```

Duration.FromText

从指定的文本 text 返回持续时间值。此函数可对以下格式进行分析:

(-)hh:mm(:ss(.ff))
(-)ddd(.hh:mm(:ss(.ff)))

(包括所有范围)

ddd: 天数。

hh: 小时数，介于 0 到 23 之间。

mm: 分钟数，介于 0 到 59 之间。

ss: 秒数，介于 0 到 59 之间。

ff: 秒的小数部分，介于 0 到 9999999 之间。

Duration.FromText(text as nullable text) as nullable duration

将 "2.05:55:20" 转换为 duration 值。

```
Duration.FromText("2.05:55:20")
#duration(2, 5, 55, 20)
```

Duration.Hours

返回所提供的 duration 值 duration 的小时部分。

Duration.Hours(duration as nullable duration) as nullable number

查找 #duration(5, 4, 3, 2) 中的小时数。

```
Duration.Hours(#duration(5, 4, 3, 2))
4
```

Duration.Minutes

返回所提供的 duration 值 duration 的分钟部分。

Duration.Minutes(duration as nullable duration) as nullable number

查找 #duration(5, 4, 3, 2) 中的分钟数。

```
Duration.Minutes(#duration(5, 4, 3, 2))
3
```

Duration.Seconds

返回所提供的 duration 值 duration 的秒部分。

Duration.Seconds(duration as nullable duration) as nullable number

查找 #duration(5, 4, 3, 2) 中的秒数。

```
Duration.Seconds(#duration(5, 4, 3, 2))
2
```

Duration.ToRecord

返回包含 duration 值 duration 的各个部分的记录。

duration : 从中创建记录的 duration 。

```
Duration.ToRecord( duration as duration) as record
```

将 #duration(2, 5, 55, 20) 转换为包括天、小时、分钟和秒这些部分的记录(如果适用)。

```
Duration.ToRecord(#duration(2, 5, 55, 20))
[Days = 2,
Hours = 5,
Minutes = 55,
Seconds = 20]
```

● Duration.ToText

返回给定 duration 值 duration 的 "day.hour:mins:sec" 格式的文本表示形式。

指定格式的文本值可以作为可选的第二个参数 format 提供。

duration : 从中计算文本表示形式的 duration 。

format : [可选] 指定格式的 text 值。

```
Duration.ToText( duration as nullable duration, optional format as nullable text) as nullable text
```

将 #duration(2, 5, 55, 20) 转换为文本值。

```
Duration.ToText(#duration(2, 5, 55, 20))
"2.05:55:20"
```

● Duration.TotalDays

返回所提供的 duration 值 duration 跨越的总天数。

```
Duration.TotalDays( duration as nullable duration) as nullable number
```

计算 #duration(5, 4, 3, 2) 中跨越的总天数。

```
Duration.TotalDays(#duration(5, 4, 3, 2))
5.1687731481481478
```

● Duration.TotalHours

返回所提供的 duration 值 duration 跨越的总小时数。

```
Duration.TotalHours( duration as nullable duration) as nullable number
```

计算 #duration(5, 4, 3, 2) 中跨越的总小时数。

```
Duration.TotalHours(#duration(5, 4, 3, 2))
```

124.05055555555555

- **Duration.TotalMinutes**

返回所提供的 duration 值 duration 跨越的总分钟数。

Duration.TotalMinutes(duration as nullable duration) as nullable number

计算 #duration(5, 4, 3, 2) 中跨越的总分钟数。

```
Duration.TotalMinutes(#duration(5, 4, 3, 2))
```

7443.0333333333338

- **Duration.TotalSeconds**

返回所提供的 duration 值 duration 跨越的总秒数。

Duration.TotalSeconds(duration as nullable duration) as nullable number

计算 #duration(5, 4, 3, 2) 中跨越的总秒数。

```
Duration.TotalSeconds(#duration(5, 4, 3, 2))
```

446582

- **Duration.Type**

表示所有 duration 值的类型

- **Embedded.Value**

在嵌入的混合资源中按名称访问值。

Embedded.Value(value as any, path as text) as any

- **Error.Record**

从为原因、消息和详细信息提供的文本值返回错误记录。

Error.Record(reason as text, optional message as nullable text, optional detail as any) as record

- **Excel.CurrentWorkbook**

返回当前 Excel 工作簿中的表。

Excel.CurrentWorkbook() as table

- **Excel.Workbook**

从 Excel 工作簿返回工作表的记录。

Excel.Workbook(workbook as binary, optional useHeaders as nullable logical, optional delayTypes as nullable logical) as table

- **Exchange.Contents**

返回 Microsoft Exchange 帐户 mailboxAddress 中的内容表。如果未指定 mailboxAddress ，则将使用凭据的默认帐户。

```
Exchange.Contents(optional mailboxAddress as nullable text) as table
```

- **Expression.Constant**

Expression.Constant

```
Expression.Constant( value as any) as text
```

- **Expression.Evaluate**

Expression.Evaluate

```
Expression.Evaluate( document as text, optional environment as nullable record) as any
```

- **Expression.Identifier**

Expression.Identifier

```
Expression.Identifier( name as text) as text
```

- **ExtraValues.Error**

如果拆分器功能返回的列比表所需的列多，应引发错误。

- **ExtraValues.Ignore**

如果拆分器功能返回的列比表所需的列多，应忽略它们。

- **ExtraValues.List**

如果拆分器功能返回的列比表所需的列多，应将它们收集到列表中。

- **ExtraValues.Type**

指定对包含的列数多于预期的行中的额外值应采取的操作。

- **Facebook.Graph**

返回一条记录，它包含在位于指定 URL url 的 Facebook 图形中找到的一组表。

```
Facebook.Graph( url as text) as any
```

- **File.Contents**

以二进制形式返回文件 path 的内容。

File.Contents(path as text) as binary

- **Folder.Contents**

返回一个表，它包含在文件夹路径 path 中找到的每个文件夹和文件所对应的行。每一行都包含所对应的文件夹或文件的属性以及指向其内容的链接。

Folder.Contents(path as text) as table

- **Folder.Files**

返回一个表，它包含在文件夹路径 path 和子文件夹中找到的每个文件所对应的行。每一行都包含所对应的文件的属性以及指向其内容的链接。

Folder.Files(path as text) as table

- **Function.Invoke**

使用指定的参数列表调用给定的函数并返回结果。

Function.Invoke(function as function, args as list) as any

使用一个参数 [A=1,B=2] 调用 Record.FieldNames

```
Function.Invoke(Record.FieldNames, {[A=1,B=2]})
{ "A", "B" }
```

- **Function.InvokeAfter**

经过持续时间 delay 后，返回调用 function 的结果。

Function.InvokeAfter(function as function, delay as duration) as any

- **Function.IsDataSource**

返回是否将 function 视为数据源。

Function.IsDataSource(function as function) as logical

- **Function.Type**

表示所有函数的类型。

- **GroupKind.Global**

GroupKind.Global

- **GroupKind.Local**

GroupKind.Local

● GroupKind.Type

指定分组类型。

Global：全局组是输入表中具有相同键值的所有行组成的。

Local：本地组是输入表中具有相同键值的连续顺序行组成的。

可能会生成具有相同键值的多个本地组，但只会为给定键值生成单个全局组。

● HdInsight.Containers

返回包含从 Azure 存储库的帐户 URL account 中找到的每个容器行的导航表。每行包含一个到容器 Blob 的链接。

HdInsight.Containers(account as text) as table

● HdInsight.Contents

返回包含从 Azure 存储库的帐户 URL account 中找到的每个容器行的导航表。每行包含一个到容器 Blob 的链接。

HdInsight.Contents(account as text) as table

● HdInsight.Files

返回包含从 Azure 存储库的容器 URL account 中找到的每个 Blob 文件行的表。

HdInsight.Files(account as text, containerName as text) as table

● Hdfs.Contents

返回一个表，它包含在 Hadoop 文件系统的文件夹 URL url 上找到的每个文件夹和文件所对应的行。每一行都包含所对应的文件夹或文件的属性以及指向其内容的链接。

Hdfs.Contents(url as text) as table

● Hdfs.Files

返回一个表，它包含在 Hadoop 文件系统的文件夹 URL url 和子文件夹中找到的每个文件所对应的行。每一行都包含所对应的文件的属性以及指向其内容的链接。

Hdfs.Files(url as text) as table

● Informix.Database

返回包含(在名为 database 的数据库实例中)服务器 server 上 Informix 数据库中可用的 SQL 表和视图的表。可以视需要使用服务器指定端口，并用冒号分隔。可以指定可选的记录参数 options 来控制以下选项:

CreateNavigationProperties :逻辑(true/false)，用于设置是否在返回的值上生成导航属性(默认值为 true)。

NavigationPropertyNameGenerator :函数，用于创建导航属性的名称。

Query :文本，用于转换为在服务器上运行的 SQL 查询。你可以指定多个查询，但系统只返回第一个结果。

CommandTimeout :持续时间，用于控制服务器端查询在取消前运行的时间。默认值为十分钟。

HierarchicalNavigation :逻辑(true/false)，用于设置是否查看按架构名称分组的表(默认值为 false)。

例如，可以将记录参数指定为 [option1 = value1, option2 = value2...] 或 [Query = "select ..."]。

Informix.Database(server as text, database as text, optional options as nullable record) as table

● Int16.From

从给定的 value 返回 16 位整数 number 值。如果给定的 value 为 null ，Int16.From 将返回 null 。如果给定的 value 为 16 位整数范围内的 number ，且没有小数部分，将返回 value 。如果有小数部分，则该数字将以指定舍入模式舍入。默认舍入模式为 RoundingMode.ToEven 。如果给定的 value 为其他类型，请参阅

Number.FromText ，将其转换为 number 值，此后有关将 number 值转换为 16 位整数 number 值的上一语句适用。有关可用的舍入模式，请参阅 Number.Round 。

Int16.From(value as any, optional culture as nullable text, optional roundingMode as nullable number) as nullable number

获取 "4" 的 16 位整数 number 值。

```
Int64.From("4")
4
----------
```

使用 RoundingMode.AwayFromZero 获取 "4.5" 的 16 位整数 number 值。

```
Int16.From("4.5", null, RoundingMode.AwayFromZero)
5
```

● Int16.Type

用于表示带符号的 16 位整数的类型。

● Int32.From

从给定的 value 返回 32 位整数 number 值。如果给定的 value 为 null ，Int32.From 将返回 null 。如果给定的 value 为 32 位整数范围内的 number ，且没有小数部分，将返回 value 。如果有小数部分，则该数字将以指定舍入模式舍入。默认舍入模式为 RoundingMode.ToEven 。如果给定的 value 为其他类型，请参阅 Number.FromText ，将其转换为 number 值，此后有关将 number 值转换为 32 位整数 number 值的上一语句适用。有关可用的舍入模式，请参阅 Number.Round 。

Int32.From(value as any, optional culture as nullable text, optional roundingMode as nullable number) as nullable number

获取 "4" 的 32 位整数 number 值。

```
Int32.From("4")
4
----------
```

使用 RoundingMode.AwayFromZero 获取 "4.5" 的 32 位整数 number 值。

```
Int32.From("4.5", null, RoundingMode.AwayFromZero)
5
```

● Int32.Type

表示 32 位有符号整数的类型。

● Int64.From

从给定的 value 返回 64 位整数 number 值。如果给定的 value 为 null ，Int64.From 将返回 null 。如果给定的 value 为 64 位整数范围内的 number ，且没有小数部分，将返回 value 。如果有小数部分，则该数字将以指定舍入模式舍入。默认舍入模式为 RoundingMode.ToEven 。如果给定的 value 为其他类型，请参阅 Number.FromText ，将其转换为 number 值，此后有关将 number 值转换为 64 位整数 number 值的上一语句适用。有关可用的舍入模式，请参阅 Number.Round 。

Int64.From(value as any, optional culture as nullable text, optional roundingMode as nullable number) as nullable number

获取 64 位整数的 number 值 "4"。

```
Int64.From("4")
4
----------
```

使用 RoundingMode.AwayFromZero 获取 "4.5" 的 64 位整数 number 值。

```
Int64.From("4.5", null, RoundingMode.AwayFromZero)
5
```

● Int64.Type

表示 64 位有符号整数的类型。

● Int8.From

从给定的 value 返回 8 位有符号整数 number 值。如果给定的 value 是 null ，Int8.From 将返回 null 。如果给定的 value 为 8 位有符号整数范围内的 number ，且没有小数部分，将返回 value 。如果有小数部分，则该数字将以指定舍入模式舍入。默认舍入模式为 RoundingMode.ToEven 。如果给定的 value 为其他类型，请参阅

Number.FromText ，将其转换为 number 值，此后有关将 number 值转换为 8 位有符号整数 number 值的上一语句适用。有关可用的舍入模式，请参阅 Number.Round 。

Int8.From(value as any, optional culture as nullable text, optional roundingMode as nullable number) as nullable number

获取 "4" 的 8 位有符号整数 number 值。

```
Int8.From("4")
4
----------
```

使用 RoundingMode.AwayFromZero 获取 "4.5" 的 8 位有符号整数 number 值。

```
Int8.From("4.5", null, RoundingMode.AwayFromZero)
5
```

- **Int8.Type**

表示 8 位有符号整数的类型。

- **JoinAlgorithm.Dynamic**

JoinAlgorithm.Dynamic

- **JoinAlgorithm.LeftHash**

JoinAlgorithm.LeftHash

- **JoinAlgorithm.LeftIndex**

JoinAlgorithm.LeftIndex

- **JoinAlgorithm.PairwiseHash**

JoinAlgorithm.PairwiseHash

- **JoinAlgorithm.RightHash**

JoinAlgorithm.RightHash

- **JoinAlgorithm.RightIndex**

JoinAlgorithm.RightIndex

- **JoinAlgorithm.SortMerge**

JoinAlgorithm.SortMerge

- **JoinAlgorithm.Type**

指定联接操作中要使用的联接算法。

- **JoinKind.FullOuter**

Table.Join 中可选 JoinKind 参数的可能值。

完整外部联接确保两个表的所有行显示在结果中。在其他表中没有匹配的行将会与在其所有列中包含 null 值的“默认”行联接起来。

- **JoinKind.Inner**

Table.Join 中可选 JoinKind 参数的可能值。

由内部联接生成的表包含着根据指定键列确定为匹配的指定表中的每个行对所对应的行。

- **JoinKind.LeftAnti**

Table.Join 中可选 JoinKind 参数的可能值。

左边的反联接返回在第二个表中没有匹配项的第一个表中的所有行。

- **JoinKind.LeftOuter**

Table.Join 中可选 JoinKind 参数的可能值。

左边的外部联接确保第一个表的所有行显示在结果中。

- **JoinKind.RightAnti**

Table.Join 中可选 JoinKind 参数的可能值。

右边的反联接返回在第一个表中没有匹配项的第二个表中的所有行。

- **JoinKind.RightOuter**

Table.Join 中可选 JoinKind 参数的可能值。

右边的外部联接确保第二个表的所有行显示在结果中。

- **JoinKind.Type**

指定联接操作的类型。

- **Json.Document**

返回 JSON 文档的内容。

Json.Document(jsonText as any, optional encoding as nullable number) as any

● Json.FromValue

使用由 encoding 指定的文本编码生成给定值 value 的 JSON 表示形式。如果省略 encoding ，则使用 UTF8。值的表示形式如下:

Null、文本和逻辑值表示为相应的 JSON 类型

数字表示为 JSON 形式的数字，除非 #infinity 、 -#infinity 和 #nan 均转换为 null

列表表示为 JSON 数组

记录表示为 JSON 对象

表表示为对象的数组

日期、时间、日期时间、日期时间时区和持续时间表示为 ISO-8601 文本

二进制值表示为 Base64 编码文本

类型和函数将生成错误

Json.FromValue(value as any, optional encoding as nullable number) as binary

将复杂值转换为 JSON。

```
Text.FromBinary(Json.FromValue([A={1, true, "3"}, B=#date(2012, 3, 25)]))
"{""A"":[1,true,""3""],""B"":""2012-03-25""}"
```

● Lines.FromBinary

将二进制值转换成在换行符处拆分的文本值列表。如果指定的是引用样式，则引号内可能会出现换行符。如果 includeLineSeparators 为 true，则文本中可能会出现换行符。

Lines.FromBinary(binary as binary, optional quoteStyle as nullable number, optional includeLineSeparators as nullable logical, optional encoding as nullable number) as list

● Lines.FromText

将文本值转换为在换行符处拆分的文本值列表。如果 includeLineSeparators 为 true，则文本中将包括换行符。

QuoteStyle.None: (默认值)无需任何加引号行为。

QuoteStyle.Csv: 如何加引号按 CSV 的要求进行。使用一个双引号字符来界定这些区域，使用一对双引号来表示此区域中的单个双引号字符。

```
Lines.FromText( text as text, optional quoteStyle as nullable number, optional includeLineSeparators as nullable logical) as list
```

- **Lines.ToBinary**

使用指定的编码和 lineSeparator 将文本列表转换为二进制值。指定的 lineSeparator 追加到每行之后。 如果未指定，则使用回车和换行符。

```
Lines.ToBinary( lines as list, optional lineSeparator as nullable text, optional encoding as nullable number, optional includeByteOrderMark as nullable logical) as binary
```

- **Lines.ToText**

将文本列表转换为单个文本。 指定的 lineSeparator 追加到每行之后。 如果未指定，则使用回车和换行符。

```
Lines.ToText( lines as list, optional lineSeparator as nullable text) as text
```

- **List.Accumulate**

使用 accumulator 从列表 list 中的项累积汇总值。

可以设置可选的种子参数 seed 。

```
List.Accumulate( list as list, seed as any, accumulator as function) as any
```

```
使用 ((state, current) => state + current ) 从列表 {1, 2, 3, 4, 5} 中的项累积汇总值。

List.Accumulate({1, 2, 3, 4, 5}, 0, (state, current) => state + current)
15
```

- **List.AllTrue**

如果列表 list 中的所有表达式均为 true，则返回 true。

```
List.AllTrue( list as list) as logical
```

```
确定列表 {true, true, 2 > 0} 中的所有表达式是否均为 true。

List.AllTrue({true, true, 2 > 0})
true
----------
```

确定列表 {true, true, 2 < 0} 中的所有表达式是否均为 true。

```
List.AllTrue({true, false, 2 < 0})
false
```

List.Alternate

返回由列表中所有奇数编号的偏移量元素组成的列表。根据参数在从列表 list 取值和跳过其值间切换。

count：指定每次跳过的值数。

repeatInterval：可选的重复间隔，指示在两个跳过的值之间添加了多少个值。

offset：一个可选偏移量参数，指示在初始偏移量处开始跳过值。

```
List.Alternate( list as list, count as number, optional repeatInterval as nullable number, optional offset as nullable number) as list
```

从 {1..10} 创建跳过第一个数的列表。

```
List.Alternate({1..10}, 1)
{2, 3, 4, 5, 6, 7, 8, 9, 10}
----------
```

从 {1..10} 创建每隔一个数跳过数的列表。

```
List.Alternate({1..10}, 1, 1)
{2, 4, 6, 8, 10}
----------
```

从 {1..10} 创建从 1 开始、每隔一个数跳过数的列表。

```
List.Alternate({1..10}, 1, 1, 1)
{1, 3, 5, 7, 9}
----------
```

从 {1..10} 创建从 1 开始先跳过一个值、接着保留两个值这样依次进行得到的列表。

```
List.Alternate({1..10}, 1, 2, 1)
{1, 3, 4, 6, 7, 9, 10}
```

List.AnyTrue

如果列表 list 中的任意表达式为 true，则返回 true。

```
List.AnyTrue( list as list) as logical
```

确定列表 {true, false, 2 > 0} 中的任意表达式是否为 true。

List.AnyTrue({true, false, 2>0})
true

确定列表 {2 = 0, false, 2 < 0} 中的任意表达式是否为 true。

List.AnyTrue({2 = 0, false, 2 < 0})
false

List.Average

返回列表 list 中项的平均值。采用与列表中的值所属的同一数据类型给出结果。仅处理 number、date、time、datetime、datetimezone 和 duration 值。

如果列表为空，则返回 null。

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

计算数的列表 {3, 4, 6} 的平均值。

List.Average({3, 4, 6})
4.333333333333333

计算 date 值 2011 年 1 月 1 日、2011 年 1 月 2 日和 2011 年 1 月 3 日的平均值。

List.Average({#date(2011, 1, 1), #date(2011, 1, 2), #date(2011, 1, 3)})
#date(2011, 1, 2)

List.Buffer

在内存中缓冲列表 list 。此调用的结果是一个稳定的列表。

List.Buffer(list as list) as list

创建列表 {1..10} 的稳定副本。

List.Buffer({1..10})
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

List.Combine

提取列表的列表 lists 并将它们合并为一个新列表。

List.Combine(lists as list) as list

合并两个简单的列表 {1, 2} 和 {3, 4}。

List.Combine({{1, 2}, {3, 4}})
{
1,

2,
3,
4
}

合并两个列表 {1, 2} 和 {3, {4, 5}}，其中一个包含嵌套的列表。

List.Combine({{1, 2}, {3, {4, 5}}})
{
1,
2,
3, {4,
5}
}

● List.Contains

指示列表 list 是否包含值 value 。

如果在列表中找到值，则为 true；否则为 false。可以指定一个可选相等条件值

equationCriteria 来控制相等测试。

List.Contains(list as list, value as any, optional equationCriteria as any) as logical

查看列表 {1, 2, 3, 4, 5} 是否包含 3。

List.Contains({1, 2, 3, 4, 5}, 3)
true

查看列表 {1, 2, 3, 4, 5} 是否包含 6。

List.Contains({1, 2, 3, 4, 5}, 6)
false

● List.ContainsAll

指示列表 list 是否包含另一个列表 values 中的所有值。

如果在列表中找到值，则为 true；否则为 false。可以指定一个可选相等条件值

equationCriteria 来控制相等测试。

List.ContainsAll(list as list, values as list, optional equationCriteria as any) as logical

查看列表 {1, 2, 3, 4, 5} 是否包含 3 和 4。

List.ContainsAll({1, 2, 3, 4, 5}, {3, 4})

true

查看列表 {1, 2, 3, 4, 5} 是否包含 5 和 6。

List.ContainsAll({1, 2, 3, 4, 5}, {5, 6})
false

- ## List.ContainsAny

指示列表 list 是否包含另一个列表 values 中的任意值。

如果在列表中找到值，则为 true；否则为 false。可以指定一个可选相等条件值

equationCriteria 来控制相等测试。

List.ContainsAny(list as list, values as list, optional equationCriteria as any) as logical

查看列表 {1, 2, 3, 4, 5} 是否包含 3 或 9。

List.ContainsAny({1, 2, 3, 4, 5}, {3, 9})
true

查看列表 {1, 2, 3, 4, 5} 是否包含 6 或 7。

List.ContainsAny({1, 2, 3, 4, 5}, {6, 7})
false

- ## List.Count

返回列表 list 中的项数。

List.Count(list as list) as number

查看列表 {1, 2, 3} 中的值数。

List.Count({1, 2, 3})
3

- ## List.Covariance

返回两个列表 numberList1 和 numberList2 之间的协方差。 numberList1 和 numberList2 必须包含相同数目的 number 值。

List.Covariance(numberList1 as list, numberList2 as list) as nullable number

计算两个列表之间的协方差。

List.Covariance({1, 2, 3},{1, 2, 3})
0.66666666666666607

● List.DateTimeZones

返回大小为 count 的 datetimezone 值的列表，从 start 开始。给定的增量 step 是加到每个值的 duration 值。

List.DateTimeZones(start as datetimezone, count as number, step as duration) as list

从新年前 5 分钟(#datetimezone(2011, 12, 31, 23, 55, 0, -8, 0))开始创建 10 个值的列表，以 1 分钟为增量(#duration(0, 0, 1, 0))。

```
List.DateTimeZones(#datetimezone(2011, 12, 31, 23, 55, 0, -8, 0), 10, #duration(0, 0, 1, 0))
{
#datetimezone(2011, 12, 31, 23, 55, 0, -8, 0),
#datetimezone(2011, 12, 31, 23, 56, 0, -8, 0),
#datetimezone(2011, 12, 31, 23, 57, 0, -8, 0),
#datetimezone(2011, 12, 31, 23, 58, 0, -8, 0),
#datetimezone(2011, 12, 31, 23, 59, 0, -8, 0),
#datetimezone(2012, 1, 1, 0, 0, 0, -8, 0),
#datetimezone(2012, 1, 1, 0, 1, 0, -8, 0),
#datetimezone(2012, 1, 1, 0, 2, 0, -8, 0),
#datetimezone(2012, 1, 1, 0, 3, 0, -8, 0),
#datetimezone(2012, 1, 1, 0, 4, 0, -8, 0)
}
```

● List.DateTimes

返回大小为 count 的 datetime 值的列表，从 start 开始。给定的增量 step 是加到每个值的 duration 值。

List.DateTimes(start as datetime, count as number, step as duration) as list

从新年前 5 分钟(#datetime(2011, 12, 31, 23, 55, 0))开始创建 10 个值的列表，以 1 分钟为增量(#duration(0, 0, 1, 0))。

```
List.DateTimes(#datetime(2011, 12, 31, 23, 55, 0), 10, #duration(0, 0, 1, 0))
{
#datetime(2011, 12, 31, 23, 55, 0),
#datetime(2011, 12, 31, 23, 56, 0),
#datetime(2011, 12, 31, 23, 57, 0),
#datetime(2011, 12, 31, 23, 58, 0),
#datetime(2011, 12, 31, 23, 59, 0),
#datetime(2012, 1, 1, 0, 0, 0),
#datetime(2012, 1, 1, 0, 1, 0),
```

```
#datetime(2012, 1, 1, 0, 2, 0),
#datetime(2012, 1, 1, 0, 3, 0),
#datetime(2012, 1, 1, 0, 4, 0)
}
```

● List.Dates

返回大小为 count 的 date 值的列表，从 start 开始。给定的增量 step 是加到每个值的 duration 值。

```
List.Dates( start as date, count as number, step as duration) as list
```

创建 5 个值的列表，从新年除夕(#date(2011, 12, 31))开始，以 1 天为增量(#duration(1, 0, 0, 0))。

```
List.Dates(#date(2011, 12, 31), 5, #duration(1, 0, 0, 0))
{
#date(2011, 12, 31),
#date(2012, 1, 1),
#date(2012, 1, 2),
#date(2012, 1, 3),
#date(2012, 1, 4)
}
```

● List.Difference

返回未出现在列表 list2 中的列表 list1 中的项。支持重复的值。

可以指定一个可选相等条件值 equationCriteria 来控制相等测试。

```
List.Difference( list1 as list, list2 as list, optional equationCriteria as any) as list
```

查找列表 {1, 2, 3, 4, 5} 中未出现在 {4, 5, 3} 中的项。

```
List.Difference({1, 2, 3, 4, 5},{4, 5, 3})
{1, 2}
----------
```

查找列表 {1, 2} 中未出现在 {1, 2, 3} 中的项。

```
List.Difference({1, 2}, {1, 2, 3})
{}
```

● List.Distinct

返回包含列表 list 中的所有值且删除了重复项的列表。如果列表为空，则结果为空列表。

```
List.Distinct( list as list, optional equationCriteria as any) as list
```

从列表 {1, 1, 2, 3, 3, 3} 中删除重复的项。

List.Distinct({1, 1, 2, 3, 3, 3})
{1, 2, 3}

List.Durations

返回 count duration 值的列表，从 start 开始，以给定的 duration step 为增量。

List.Durations(start as duration, count as number, step as duration) as list

创建 5 个值的列表，从 1 小时开始，以 1 小时为增量。

List.Durations(#duration(0, 1, 0, 0), 5, #duration(0, 1, 0, 0))
{#duration(0, 1, 0, 0),
#duration(0, 2, 0, 0),
#duration(0, 3, 0, 0),
#duration(0, 4, 0, 0),
#duration(0, 5, 0, 0)}

List.FindText

从包含值 text 的列表 list 返回值列表。

List.FindText(list as list, text as text) as list

在列表 {"a", "b", "ab"} 中查找匹配 "a" 的文本值。

List.FindText({"a", "b", "ab"}, "a")
{"a", "ab"}

List.First

返回列表 list 中的第一个项；如果列表为空，则返回可选的默认值 defaultValue 。

如果列表为空且未指定默认值，函数将返回 null 。

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

查找列表 {1, 2, 3} 中的第一个值。

List.First({1, 2, 3})
1

查找列表 {} 中的第一个值。如果列表为空，则返回 -1。

List.First({}, -1)
-1

List.FirstN

通过指定要返回的项数或限定条件来返回列表中的第一组项。

如果指定一个数，则最多返回这么多项。

如果指定一个条件，则返回最初满足该条件的所有项。一旦某个项不满足该条件，则不再考虑其他项。

List.FirstN(list as list, countOrCondition as any) as any

在列表 {3, 4, 5, -1, 7, 8, 2} 中查找大于 0 的初始值。

List.FirstN({3, 4, 5, -1, 7, 8, 2},each _ > 0)
{3, 4, 5}

● List.Generate

给定生成初始值 initial 的四个函数，针对条件 condition 进行测试，如果成功，则选择结果并生成下一个值 next ，以此生成值列表。

还可以指定可选参数 selector 。

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

创建从 10 开始，大于 0 且按 1 递减的值的列表。

List.Generate(()=>10, each _ > 0, each _ - 1)
{10, 9, 8, 7, 6, 5, 4, 3, 2, 1}

生成包含 x 和 y 的记录的列表，其中 x 是一个值，y 是一个列表。x 应保持小于 10 并表示列表 y 中的项数。在生成列表后，只返回 x 值。

List.Generate(()=> [x = 1 , y = {}] , each [x] < 10 , each [x = List.Count([y]), y = [y] & {x}] , each [x])
{1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

● List.InsertRange

返回通过在 index 处将 values 中的值插入 list 来生成的新列表。列表中的第一个位置为索引 0 处。

list : 要插入值的目标列表。

index : 目标列表(list)的索引，要在此处插入值。列表中的第一个位置为索引 0 处。

values：要插入 list 的值列表。

List.InsertRange(list as list, index as number, values as list) as list

在索引 2 处将列表({3, 4})插入目标列表({1, 2, 5})。

```
List.InsertRange({1, 2, 5}, 2, {3, 4})
{
1,
2,
3,
4,
5
}
----------------
```

在索引 0 处将带嵌套列表的列表({1, {1.1, 1.2}})插入目标列表({2, 3, 4})。

```
List.InsertRange({2, 3, 4}, 0, {1, {1.1, 1.2}})
{
1, {
1.1,
1.2
},
2,
3,
4
}
```

● List.Intersect

返回在输入列表 lists 中找到的列表值的交集。可以指定可选参数 equationCriteria 。

List.Intersect(lists as list, optional equationCriteria as any) as list

查找列表 {1..5}、{2..6}、{3..7} 的交集。

```
List.Intersect({{1..5}, {2..6}, {3..7}})
{3, 4, 5}
```

● List.IsDistinct

返回一个指示列表 list 中是否有重复值的逻辑值；如果列表没有重复值，则为 true ，否则为 false 。

List.IsDistinct(list as list, optional equationCriteria as any) as logical

查看列表 {1, 2, 3} 是否具有独特性(即 没有重复值)。

List.IsDistinct({1, 2, 3})
true

查看列表 {1, 2, 3, 3} 是否具有独特性(即 没有重复值)。

List.IsDistinct({1, 2, 3, 3})
false

- **List.IsEmpty**

如果列表 list 不包含值(长度为 0)，则返回 true 。如果列表包含值(长度 > 0)，则返回 false 。

List.IsEmpty(list as list) as logical

查看列表 {} 是否为空。

List.IsEmpty({})
true

查看列表 {1, 2} 是否为空。

List.IsEmpty({1, 2})
false

- **List.Last**

返回列表 list 中的最后一项；如果列表为空，则返回可选的默认值 defaultValue 。如果列表为空且未指定默认值，函数将返回 null 。

List.Last(list as list, optional defaultValue as any) as any

查找列表 {1, 2, 3} 中的最后一个值。

List.Last({1, 2, 3})
3

查找列表 {} 中的最后一个值，如果列表为空，则返回 -1。

List.Last({}, -1)
-1

- **List.LastN**

返回列表 list 的最后一项。如果列表为空，将引发异常。

此函数采用一个可选参数 countOrCondition 来支持收集多个项或筛选项。可以通过以下三种方式指定 countOrCondition :

如果指定一个数，则最多返回这么多项。

如果指定一个条件，则返回最初满足该条件的所有项，从列表末尾开始。一旦某个项不满足该条件，则不再考虑其他项。

如果此参数为 null，则返回列表中的最后一项。

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

```
查找列表 {3, 4, 5, -1, 7, 8, 2} 中的最后一个值。

List.LastN({3, 4, 5, -1, 7, 8, 2},1)
{2}
----------

查找列表 {3, 4, 5, -1, 7, 8, 2} 中大于 0 的最后一个值。

List.LastN({3, 4, 5, -1, 7, 8, 2}, each _ > 0)
{7, 8, 2}
```

● List.MatchesAll

如果列表 list 中的所有值均满足条件函数 condition ，则返回 true ；否则返回 false 。

```
List.MatchesAll( list as list, condition as function) as logical
```

```
确定列表 {11, 12, 13} 中的所有值是否大于 10。

List.MatchesAll({11, 12, 13},each _ > 10)
true
----------

确定列表 {1, 2, 3} 中的所有值是否大于 10。

List.MatchesAll({1, 2, 3},each _ > 10)
false
```

● List.MatchesAny

如果列表 list 中的任意值满足条件函数 condition ，则返回 true ；否则返回 false 。

```
List.MatchesAny( list as list, condition as function) as logical
```

```
查看列表 {9, 10, 11} 中的任意值是否大于 10。

List.MatchesAny({9, 10, 11},each _ > 10)
```

true

查看列表 {1, 2, 3} 中的任意值是否大于 10。

List.MatchesAny({1, 2, 3},each _ > 10)
false

- **List.Max**

返回列表 list 中的最大项；如果列表为空，则返回可选的默认值 default 。

可以指定可选的 comparisonCriteria 值 comparisonCriteria 来确定如何在列表中比较项。如果此参数为 null，将使用默认比较器。

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

在列表 {1, 4, 7, 3, -2, 5} 中查找最大值。

List.Max({1, 4, 7, 3, -2, 5},1)
7

查找列表 {} 中的最大值，如果列表为空，则返回 -1。

List.Max({}, -1)
-1

- **List.MaxN**

返回列表 list 中的最大值。

在对行排序后，可以指定可选参数以进一步筛选结果。可选参数 countOrCondition 指定要返回的值数或筛选条件。可选参数 comparisonCriteria 指定如何比较列表中的值。

list : 值的列表。

countOrCondition : 如果指定一个数，则返回以升序排序的、最多包含 countOrCondition 项的列表。如果指定条件，则返回最初满足该条件的项列表。一旦某个项不满足该条件，则不再考虑其他项。

comparisonCriteria : [Opional] 可以指定可选的 comparisonCriteria 值来确定如何比较列表中的项。如果此参数为 null，将使用默认比较器。

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

List.Median

返回列表 list 的中位数项。如果列表为空，此函数将引发异常。

如果具有偶数项，则函数选择两个中位数中的较小者。

List.Median(list as list, optional comparisonCriteria as any) as any

查找列表 {5, 3, 1, 7, 9} 的中位数。

```
List.Median({5, 3, 1, 7, 9})
5
```

List.Min

返回列表 list 中的最小项；如果列表为空，则返回可选的默认值 default 。

可以指定可选的 comparisonCriteria 值 comparisonCriteria 来确定如何在列表中比较项。如果此参数为 null，将使用默认比较器。

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

查找列表 {1, 4, 7, 3, -2, 5} 中的最小值。

```
List.Min({1, 4, 7, 3, -2, 5})
-2
----------
```

查找列表 {} 中的最小值，如果列表为空，则返回 -1。

```
List.Min({}, -1)
-1
```

List.MinN

返回列表 list 中的最小值。

参数 countOrCondition 指定要返回的值数或筛选条件。可选参数 comparisonCriteria 指定如何比较列表中的值。

list：值的列表。

countOrCondition：如果指定一个数，则返回以升序排序的、最多包含 countOrCondition 项的列表。如果指定条件，则返回最初满足该条件的项列表。一旦某个项不满足该条件，则不再考虑其他项。如果此参数为 null，则返回列表中的一个最小值。

comparisonCriteria : [Opional] 可以指定可选的 comparisonCriteria 值来确定如何比较列表中的项。如果此参数为 null，将使用默认比较器。

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

查找列表 {3, 4, 5, -1, 7, 8, 2} 中的 5 个最小值。

List.MinN({3, 4, 5, -1, 7, 8, 2}, 5)
{-1, 2, 3, 4, 5}

● List.Mode

返回列表 list 中出现最多的项。如果列表为空，将引发异常。如果出现最多的项有多个，则选择其中的最后一项。

可以指定可选的 comparisonCriteria 值 equationCriteria 来控制相等测试。

List.Mode(list as list, optional equationCriteria as any) as any

查找列表 {"A", 1, 2, 3, 3, 4, 5} 中出现最多的项。

List.Mode({"A", 1, 2, 3, 3, 4, 5})
3

查找列表 {"A", 1, 2, 3, 3, 4, 5, 5} 中出现最多的项。

List.Mode({"A", 1, 2, 3, 3, 4, 5, 5})
5

● List.Modes

返回列表 list 中出现最多的项。如果列表为空，将引发异常。如果出现最多的项有多个，则选择其中的最后一项。

可以指定可选的 comparisonCriteria 值 equationCriteria 来控制相等测试。

```
List.Modes( list as list, optional equationCriteria as any) as list
```

查找列表 {"A", 1, 2, 3, 3, 4, 5, 5} 中出现最多的项。

```
List.Modes({"A", 1, 2, 3, 3, 4, 5, 5})
{3, 5}
```

● List.NonNullCount

返回列表 list 中的非 null 项数。

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

● List.Numbers

给定初始值、计数和可选的增量值来返回数的列表。默认增量值为 1。

start : 列表中的初始值。

count : 要创建的值数。

increment : [可选] 要按其递增的值。如果省略，则按 1 递增值。

```
List.Numbers( start as number, count as number, optional increment as nullable number) as list
```

生成从 1 开始的 10 个连续数的列表。

```
List.Numbers(1, 10)
{
1,
2,
3,
4,
5,
6,
7,
8,
9,
10
}
----------
```

生成从 1 开始的 10 个数的列表，每个后续数按 2 递增。

```
List.Numbers(1, 10, 2)
{
1,
3,
5,
7,
9,
11,
13,
15,
17,
19
}
```

● List.PositionOf

返回值 value 在列表 list 中出现时的偏移量。如果值未出现，则返回 -1。

可以指定可选的出现次数参数 occurrence 。

occurrence：要报告的最大出现次数。

```
List.PositionOf( list as list, value as any, optional occurrence as nullable number, optional equationCriteria as any) as any
```

查找列表 {1, 2, 3} 中出现值 3 的位置。

```
List.PositionOf({1, 2, 3}, 3)
2
```

● List.PositionOfAny

返回值在列表 values 中第一次出现的位置列表 list 中的偏移量。如果找不到该值，则返回 -1。

可以指定可选的出现次数参数 occurrence 。

occurrence：可以返回的最大出现次数。

```
List.PositionOfAny( list as list, values as list, optional occurrence as nullable number, optional equationCriteria as any) as any
```

查找列表 {1, 2, 3} 中第一次出现值 2 或 3 的位置。

```
List.PositionOfAny({1, 2, 3}, {2, 3})
```

1

List.Positions

返回输入列表 list 的偏移量列表。

使用 List.Transform 更改列表时，可以使用位置列表来授予对位置的转换权限。

List.Positions(list as list) as list

查找列表 {1, 2, 3, 4, null, 5} 中值的偏移量。

List.Positions({1, 2, 3, 4, null, 5})
{0, 1, 2, 3, 4, 5}

List.Product

返回列表 numbersList 中的非 null 数的乘积。如果列表中没有非 null 值，则返回 null。

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

计算列表 {1, 2, 3, 3, 4, 5, 5} 中的数的乘积。

List.Product({1, 2, 3, 3, 4, 5, 5})
1800

List.Random

给定要生成的值数和一个可选种子值，以此返回随机数(介于 0 到 1 之间)的列表。

count :要生成的随机值数。

seed : [可选] 用于作为随机数生成器种子的数值。如果省略，则在每次调用函数时生成随机数的唯一列表。如果指定带数字的种子值，则每次调用函数时，生成随机数的相同列表。

List.Random(count as number, optional seed as nullable number) as list

创建 3 个随机数的列表。

List.Random(3)
{0.992332, 0.132334, 0.023592}

创建 3 个随机数的列表，指定种子值。

List.Random(3, 2)
{0.883002, 0.245344, 0.723212}

List.Range

返回从偏移量 list 开始的列表的子集。可选参数 offset 用于设置子集中的最大项数。

List.Range(list as list, offset as number, optional count as nullable number) as list

查找包含数字 1-10 的列表中从偏移量 6 开始的子集。

```
List.Range({1..10}, 6)
{7, 8, 9, 10}
----------
```

查找包含数字 1-10 的列表中从偏移量 6 开始、长度为 2 的子集。

```
List.Range({1..10}, 6, 2)
{7, 8}
```

● List.RemoveFirstN

返回删除列表 list 的第一个元素的列表。如果 list 为空列表，则返回空列表。

此函数取一个可选参数 countOrCondition 来支持删除以下所列的多个值。

如果指定一个数字，则最多删除这么多项。

如果指定条件，则返回的列表以 list 中满足条件的第一个元素开头。一旦某个项不满足该条件，则不再考虑其他项。

如果此参数为 null，将采用默认行为。

List.RemoveFirstN(list as list, optional countOrCondition as any) as list

从 {1, 2, 3, 4, 5} 创建不带前 3 个数的列表。

```
List.RemoveFirstN({1, 2, 3, 4, 5}, 3)
{4, 5}
----------
```

从 {5, 4, 2, 6, 1} 创建一个列表，它以小于 3 的数开头。

```
List.RemoveFirstN({5, 4, 2, 6, 1}, each _ > 3)
{2, 6, 1}
```

● List.RemoveItems

从 list1 中删除 list2 中所有出现的给定值。如果 list2 中的值在 list1 中不存在，则返回原始列表。

List.RemoveItems(list1 as list, list2 as list) as list

从列表 {1, 2, 3, 4, 2, 5, 5} 中删除在列表 {2, 4, 6} 中出现的项。

```
List.RemoveItems({1, 2, 3, 4, 2, 5, 5}, {2, 4, 6})
{1, 3, 5, 5}
```

List.RemoveLastN

返回一个列表，它从列表 list 末尾删除最后 countOrCondition 个元素。如果 list 中的元素少于 countOrCondition 个，则返回空列表。

如果指定一个数字，则最多删除这么多项。

如果指定条件，则返回的列表以 list 中满足条件的倒数第一个元素结尾。一旦某个项不满足该条件，则不再考虑其他项。

如果此参数为 null，则仅删除一项。

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

从 {1, 2, 3, 4, 5} 创建不带后 3 个数的列表。

```
List.RemoveLastN({1, 2, 3, 4, 5}, 3)
{1, 2}
----------
```

从 {5, 4, 2, 6, 4} 创建一个列表，它以小于 3 的数结尾。

```
List.RemoveLastN({5, 4, 2, 6, 4}, each _ > 3)
{5, 4, 2}
```

List.RemoveMatchingItems

从列表 list1 中删除 list2 中所有出现的给定值。如果 list2 中的值在 list1 中不存在，则返回原始列表。

可以指定一个可选相等条件值 equationCriteria 来控制相等测试。

```
List.RemoveMatchingItems( list1 as list, list2 as list, optional equationCriteria as any) as list
```

从 {1, 2, 3, 4, 5, 5} 创建一个不包含 {1, 5} 的列表。

```
List.RemoveMatchingItems({1, 2, 3, 4, 5, 5}, {1, 5})
{2, 3, 4}
```

List.RemoveNulls

在 list 中删除所有出现的 "null" 值。如果列表中没有 "null" 值，则返回原始列表。

List.RemoveNulls(list as list) as list

从列表 {1, 2, 3, null, 4, 5, null, 6} 中删除 "null" 值。

List.RemoveNulls({1, 2, 3, null, 4, 5, null, 6})
{1, 2, 3, 4, 5, 6}

- **List.RemoveRange**

在 list 中删除从指定的位置 index 开始的 count 个值。

List.RemoveRange(list as list, index as number, optional count as nullable number) as list

在列表 {1, 2, 3, 4, -6, -2, -1, 5} 中删除从索引 4 开始的 3 个值。

List.RemoveRange({1, 2, 3, 4, -6, -2, -1, 5}, 4, 3)
{1, 2, 3, 4, 5}

- **List.Repeat**

返回作为原始列表 list 的 count 次重复的列表。

List.Repeat(list as list, count as number) as list

创建将 {1, 2} 重复 3 次得到的列表。

List.Repeat({1, 2}, 3)
{1, 2, 1, 2, 1, 2}

- **List.ReplaceMatchingItems**

执行对列表 list 的指定的替换。一个替换操作 replacements 由两个值的列表、列表中提供的旧值和新值组成。

可以指定一个可选相等条件值 equationCriteria 来控制相等测试。

List.ReplaceMatchingItems(list as list, replacements as list, optional equationCriteria as any) as list

从 {1, 2, 3, 4, 5} 创建一个列表，它将值 5 替换为 -5，将值 1 替换为 -1。

List.ReplaceMatchingItems({1, 2, 3, 4, 5}, {{5, -5}, {1, -1}})
{-1, 2, 3, 4, -5}

- **List.ReplaceRange**

从指定的位置 index 开始使用列表 replaceWith 替换 list 中的 count 个值。

List.ReplaceRange(list as list, index as number, count as number, replaceWith as list) as list

使用 {3, 4} 替换列表 {1, 2, 7, 8, 9, 5} 中的 {7, 8, 9}。

```
List.ReplaceRange({1, 2, 7, 8, 9, 5}, 2, 3, {3, 4})
{1, 2, 3, 4, 5}
```

List.ReplaceValue

在值列表 list 中搜索值 oldValue ，每次找到后使用替换值 newValue 替换它。

```
List.ReplaceValue( list as list, oldValue as any, newValue as any, replacer as function) as list
```

使用 "A" 替换列表 {"a", "B", "a", "a"} 中的所有 "a" 值。

```
List.ReplaceValue({"a", "B", "a", "a"}, "a", "A", Replacer.ReplaceText)
{"A", "B", "A", "A"}
```

List.Reverse

返回将列表 list 中的值反向排序得到的列表。

```
List.Reverse( list as list) as list
```

通过将 {1..10} 反向排序创建一个列表。

```
List.Reverse({1..10})
{10, 9, 8, 7, 6, 5, 4, 3, 2, 1}
```

List.Select

从列表 list 返回匹配选择条件 selection 的值列表。

```
List.Select( list as list, selection as function) as list
```

查找列表 {1, -3, 4, 9, -2} 中大于 0 的值。

```
List.Select({1, -3, 4, 9, -2}, each _ > 0)
{1, 4, 9}
```

List.Single

如果列表 list 中只有一项，则返回该项。

如果列表中有多项或列表为空，函数将引发异常。

```
List.Single( list as list) as any
```

在列表 {1} 中查找单个值。

```
List.Single({1})
1
----------
```

在列表 {1, 2, 3} 中查找单个值。

```
List.Single({1, 2, 3})
[Expression.Error] There were too many elements in the enumeration to complete the operation.
```

List.SingleOrDefault

如果列表 list 中只有一项，则返回该项。

如果列表为空，函数将返回 null，除非指定了可选的 default 。如果列表中有多项，函数将返回错误。

```
List.SingleOrDefault( list as list, optional default as any) as any
```

在列表 {1} 中查找单个值。

```
List.SingleOrDefault({1})
1
----------
```

在列表 {} 中查找单个值。

```
List.SingleOrDefault({})
null
----------
```

在列表 {} 中查找单个值。如果为空，则返回 -1。

```
List.SingleOrDefault({}, -1)
-1
```

List.Skip

返回跳过列表 list 的第一个元素的列表。如果 list 为空列表，则返回空列表。

此函数取一个可选参数 countOrCondition 来支持跳过以下所列的多个值。

如果指定一个数，则最多跳过这么多项。

如果指定条件，则返回的列表以 list 中满足条件的第一个元素开头。一旦某个项不满足该条件，则不再考虑其他项。

如果此参数为 null，将采用默认行为。

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

从 {1, 2, 3, 4, 5} 创建不带前 3 个数的列表。

List.Skip({1, 2, 3, 4, 5}, 3)
{4, 5}

从 {5, 4, 2, 6, 1} 创建一个列表，它以小于 3 的数开头。

List.Skip({5, 4, 2, 6, 1}, each _ > 3)
{2, 6, 1}

● List.Sort

根据指定的可选条件对数据列表 list 排序。

可以指定可选参数 comparisonCriteria 作为比较条件。这可能取以下值:

要控制顺序，比较条件可以是 Order 枚举值。(Order.Descending， Order.Ascending)。

要计算用于排序的键，可以使用包含 1 个参数的函数。

要选择一个键和控制顺序，比较条件可以为包含键和顺序的列表（{each 1 / _, Order.Descending})。

要完全控制比较，可以使用包含 2 个参数的函数，它根据左边的输入和右边的输入之间的关系返回 -1、0 或 1。Value.Compare 是可用于委托此逻辑的方法。

List.Sort(list as list, optional comparisonCriteria as any) as list

将列表 {2, 3, 1} 排序。

List.Sort({2, 3, 1})
{1, 2, 3}

按降序将列表 {2, 3, 1} 排序。

List.Sort({2, 3, 1}, Order.Descending)
{3, 2, 1}

使用 Value.Compare 方法按降序将列表 {2, 3, 1} 排序。

List.Sort({2, 3, 1}, (x, y) => Value.Compare(1/x, 1/y))
{3, 2, 1}

● List.StandardDeviation

返回基于样本估计的列表 numbersList 中的值的标准偏差。

如果 numbersList 为数字列表，则返回数字。

如果为空列表或类型不属于 number 的项的列表，将引发异常。

```
List.StandardDeviation( numbersList as list) as nullable number

    计算 1 到 5 的数的标准偏差。

    List.StandardDeviation({1..5})
    1.5811388300841898
```

List.Sum

返回列表 list 中所有非 null 值的总和。 如果列表中没有非 null 值，则返回 null。

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

    计算列表 {1, 2, 3} 中的数的总和。

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

List.Times

返回大小为 count 的 time 值的列表，从 start 开始。给定的增量 step 是加到每个值的 duration 值。

```
List.Times( start as time, count as number, step as duration) as list

    从中午(#time(12, 0, 0))开始创建 4 个值的列表，以 1 小时为增量(#duration(0, 1, 0, 0))。

    List.Times(#time(12, 0, 0), 4, #duration(0, 1, 0, 0))
    {
    #time(12, 0, 0),
    #time(13, 0, 0),
    #time(14, 0, 0),
    #time(15, 0, 0)
    }
```

List.Transform

通过将转换函数 transform 应用到列表 list 来返回值的新列表。

```
List.Transform( list as list, transform as function) as list

    将 1 与列表 {1, 2} 中的每个值相加。

    List.Transform({1, 2}, each _ + 1)
    {2, 3}
```

List.TransformMany

返回一个列表，其元素是从输入列表投影而来。将 collectionTransform 函数应用到每个元素，且调用 resultTransform 函数来构造结果列表。

collectionSelector 具有签名(x as Any) => ... 其中 x 是列表中的元素。

resultTransform 投影结果的形状并具有签名(x as Any, y as Any) => ... 其中 x 是列表中的元素，y 是通过将 collectionTransform 应用到该元素获得的元素。

List.TransformMany(list as list, collectionTransform as function, resultTransform as function) as list

List.Type

表示所有列表的类型。

List.Union

取列表的列表 lists ，合并各个列表中的项，然后在输出列表中返回它们。因此，返回的列表包含所有输入列表中的所有项。

此操作维护传统的包语义，因此重复值作为 Union 的一部分匹配。

可以指定一个可选相等条件值 equationCriteria 来控制相等测试。

List.Union(lists as list, optional equationCriteria as any) as list

创建列表 {1..5}、{2..6}、{3..7} 的并集。

```
List.Union({ {1..5}, {2..6}, {3..7} })
{1, 2, 3, 4, 5, 6, 7}
```

List.Zip

提取列表的列表 lists ，并通过在同一位置合并项返回列表的列表。

List.Zip(lists as list) as list

压缩两个简单列表 {1, 2} 和 {3, 4}。

```
List.Zip({{1, 2}, {3, 4}})
{
{ 1, 3 },
{ 2, 4 }
```

```
}
----------
```

压缩两个具有不同长度 {1, 2} 和 {3} 的简单列表。

```
List.Zip({{1, 2}, {3}})
{
{ 1, 3 },
{ 2, null }
}
```

● Logical.From

从给定的 value 返回 logical 值。如果给定的 value 是 null ， Logical.From 将返回 null 。 如果给定的 value 是 logical ，则返回 value 。以下类型的值可以转换为 logical 值:

text : 来自文本值("true" 或 "false")的 logical 值。有关详细信息，请参阅 Logical.FromText 。

number : 如果 value 等于 0 ，则为 false ；否则为 true 。

如果 value 属于任何其他类型，则返回错误。

Logical.From(value as any) as nullable logical

将 2 转换为 logical 值。

```
Logical.From(2)
true
```

● Logical.FromText

从文本值 text ("true" 或 "false")创建逻辑值。如果 text 包含其他字符串，将引发异常。文本值 text 不区分大小写。

Logical.FromText(text as nullable text) as nullable logical

从文本字符串 "true" 创建逻辑值。

```
Logical.FromText("true")
true
----------
```

从文本字符串 "a" 创建逻辑值。

```
Logical.FromText("a")
[Expression.Error] Could not convert to a logical.
```

- **Logical.ToText**

从逻辑值 logicalValue (true 或 false)创建文本值。如果 logicalValue 不是逻辑值，将引发异常。

```
Logical.ToText( logicalValue  as nullable logical) as nullable text
```

```
从逻辑 true 创建一个文本值。

Logical.ToText(true)
"true"
```

- **Logical.Type**

表示所有逻辑值的类型。

- **Marketplace.Subscriptions**

从 Microsoft Azure Marketplace 返回所有已订阅的源。

```
Marketplace.Subscriptions() as table
```

- **MissingField.Error**

记录和表函数中的一个可选参数，指示缺失字段应导致错误。(这是默认参数值。)

- **MissingField.Ignore**

记录和表函数中的一个可选参数，指示应忽略缺失字段。

- **MissingField.Type**

指定对包含的列数少于预期的行中缺少的值应采取的操作。

- **MissingField.UseNull**

记录和表函数中的一个可选参数，指示缺失字段应作为 null 值包含。

- **MySQL.Database**

返回包含(在名为 database 的数据库实例中)服务器 server 上 MySQL 数据库中可用的 SQL 表、视图和存储标量函数的表。可以视需要使用服务器指定端口，并用冒号分隔。可以指

定可选的记录参数 options 来控制以下选项:

CreateNavigationProperties :逻辑(true/false)，用于设置是否在返回的值上生成导航属性(默认值为 true)。

NavigationPropertyNameGenerator :函数，用于创建导航属性的名称。

Query :文本，用于转换为在服务器上运行的 SQL 查询。你可以指定多个查询，但系统只返回第一个结果。

CommandTimeout :持续时间，用于控制服务器端查询在取消前运行的时间。默认值为十分钟。

TreatTinyAsBoolean :逻辑(true/false)，用于确定是否将服务器上的 tinyint 列强制设置为逻辑值。默认值为 true。

OldGuids :逻辑(true/false)，用于设置是将 char(36) 列(如果为 false)还是将 binary(16) 列(如果为 true)视为 GUID。默认值为 false。

ReturnSingleDatabase :逻辑(true/false)，用于设置是返回所有数据库的所有表(如果为 false)，还是返回指定数据库的表和视图(如果为 true)。默认值为 false。

HierarchicalNavigation :逻辑(true/false)，用于设置是否查看按架构名称分组的表(默认值为 false)。

例如，可以将记录参数指定为 [option1 = value1, opti...

```
MySQL.Database( server as text, database as text, optional options as nullable record) as table
```

- **None.Type**

None.Type

- **Null.Type**

表示 null 的类型。

- **Number.Abs**

返回 number 的绝对值。如果 number 为 null，则 Number.Abs 返回 null。

number：要计算其绝对值的 number 。

```
Number.Abs( number as nullable number) as nullable number
```

-3 的绝对值。

```
Number.Abs(-3)
3
```

- **Number.Acos**

返回 number 的反余弦。

```
Number.Acos( number as nullable number) as nullable number
```

- **Number.Asin**

返回 number 的反正弦。

```
Number.Asin( number as nullable number) as nullable number
```

- **Number.Atan**

返回 number 的反正切。

```
Number.Atan( number as nullable number) as nullable number
```

- **Number.Atan2**

返回两个数 y 和 x 相除的反正切。除法被构造为 y / x 。

```
Number.Atan2( y as nullable number, x as nullable number) as nullable number
```

- **Number.BitwiseAnd**

返回对 number1 和 number2 执行按位 "And" 运算所得的结果。

```
Number.BitwiseAnd( number1 as nullable number, number2 as nullable number) as nullable number
```

- **Number.BitwiseNot**

返回对 number 执行按位 "Not" 运算所得的结果。

```
Number.BitwiseNot( number as any) as any
```

- **Number.BitwiseOr**

返回对 number1 和 number2 执行按位 "Or" 所得的结果。

```
Number.BitwiseOr( number1 as nullable number, number2 as nullable number) as nullable number
```

Number.BitwiseShiftLeft

返回对 number1 执行按位左移指定的位数 number2 所得的结果。

Number.BitwiseShiftLeft(number1 as nullable number, number2 as nullable number) as nullable number

Number.BitwiseShiftRight

返回对 number1 执行按位右移指定的位数 number2 所得的结果。

Number.BitwiseShiftRight(number1 as nullable number, number2 as nullable number) as nullable number

Number.BitwiseXor

返回对 number1 和 number2 执行按位 "XOR" (异或)所得的结果。

Number.BitwiseXor(number1 as nullable number, number2 as nullable number) as nullable number

Number.Combinations

从项列表 setSize 返回具有指定组合大小 combinationSize 的唯一组合数目。

setSize：列表中的项数。

combinationSize：每个组合中的项数。

Number.Combinations(setSize as nullable number, combinationSize as nullable number) as nullable number

当每个组合为 3 个一组时从总共 5 项中计算组合数。

```
Number.Combinations(5, 3)
10
```

Number.Cos

返回 number 的余弦。

Number.Cos(number as nullable number) as nullable number

计算角 0 的余弦。

```
Number.Cos(0)
1
```

Number.Cosh

返回 number 的双曲余弦。

Number.Cosh(number as nullable number) as nullable number

● Number.E

表示 2.7182818284590451 的常量，e 的值最多取 16 位小数。

● Number.Epsilon

表示浮点数可容纳的最小正数的常量值。

● Number.Exp

返回计算 e 的 number 次幂(指数函数)所得的结果。

number：要计算其指数函数的 number 。如果 number 为 null，则 Number.Exp 返回 null。

Number.Exp(number as nullable number) as nullable number

计算 e 的 3 次幂。

```
Number.Exp(3)
20.085536923187668
```

● Number.Factorial

返回数 number 的阶乘。

Number.Factorial(number as nullable number) as nullable number

计算 10 的阶乘。

```
Number.Factorial(10)
3628800
```

● Number.From

从给定的 value 返回 number 值。如果给定的 value 是 null ， Number.From 将返回 null 。 如果给定的 value 是 number ，则返回 value 。以下类型的值可以转换为 number 值:

text：文本表示形式的 number 值。处理通用文本格式("15", "3,423.10", "5.0E-10")。有关详细信息，请参阅 Number.FromText 。

logical：对于 true 为 1，对于 false 为 0。

datetime：一个包含等效的 OLE 自动化日期的双精度浮点数。

datetimezone：一个包含与 value 的当地日期和时间等效的 OLE 自动化日期的双精度浮点数。

date：一个包含等效的 OLE 自动化日期的双精度浮点数。

time：用天的小数形式表示。

duration：用整数天和天的小数形式表示。

如果 value 属于任何其他类型，则返回错误。

```
Number.From( value  as any, optional  culture  as nullable text) as nullable number
```

```
获取 "4" 的 number 值。

Number.From("4")
4
----------

获取 #datetime(2020, 3, 20, 6, 0, 0) 的 number 值。

Number.From(#datetime(2020, 3, 20, 6, 0, 0))
43910.25
----------

获取 "12.3%" 的数值。

Number.From("12.3%")
0.123
```

● Number.FromText

从给定的文本值 text 返回 number 值。

text：数值的文本表示形式。表示形式必须采用通用数字格式 - "15"、"3,423.10"、"5.0E-10"。

```
Number.FromText( text  as nullable text, optional  culture  as nullable text) as nullable number
```

```
获取 "4" 的数值。

Number.FromText("4")
4
```

获取 "5.0e-10" 的数值。

```
Number.FromText("5.0e-10")
```

```
5E-10
```

● Number.IntegerDivide

返回一个数 number1 除以另一个数 number2 所得结果的整数部分。

如果 number1 或 number2 为 null，则 Number.IntegerDivide 返回 null。

number1：被除数。

number2：除数。

```
Number.IntegerDivide( number1 as nullable number, number2 as nullable number, optional precision as nullable number) as nullable number
```

6 除以 4。

```
Number.IntegerDivide(6, 4)
```

```
1
```

8.3 除以 3。

```
Number.IntegerDivide(8.3, 3)
```

```
2
```

● Number.IsEven

通过以下方式指示值 number 是否为偶数: 如果为偶数，则返回 true ；否则返回 false 。

```
Number.IsEven( number as number) as logical
```

检查 625 是否为偶数。

```
Number.IsEven(625)
```

```
false
```

检查 82 是否为偶数。

```
Number.IsEven(82)
```

```
true
```

● Number.IsNaN

指示值是否为 NaN (不是数字)。如果 number 与 Number.IsNaN 等效，则返回 true ；否则返回 false 。

```
Number.IsNaN( number  as number) as logical
```

检查 0 除以 0 是否为 NaN。

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

```
true
```

检查 1 除以 0 是否为 NaN。

```
Number.IsNaN(1/0)
```

```
false
```

● Number.IsOdd

指示值是否为奇数。如果 number 为奇数，则返回 true ；否则返回 false 。

```
Number.IsOdd( number  as number) as logical
```

检查 625 是否为奇数。

```
Number.IsOdd(625)
```

```
true
```

检查 82 是否为奇数。

```
Number.IsOdd(82)
```

```
false
```

● Number.Ln

返回数 number 的自然对数。如果 number 为 null，则 Number.Ln 返回 null。

```
Number.Ln( number  as nullable number) as nullable number
```

获取 15 的自然对数。

```
Number.Ln(15)
```

```
2.70805020110221
```

● Number.Log

根据指定底数 base 返回数值 number 的对数。如果未指定 base ，则默认值为 Number.E。

如果 number 为 null，则 Number.Log 返回 null。

Number.Log(number as nullable number, optional base as nullable number) as nullable number

获取 2 的以 10 为底的对数。

```
Number.Log(2, 10)
0.3010299956639812
----------
```

获取 2 的以 e 为底的对数。

```
Number.Log(2)
0.69314718055994529
```

● Number.Log10

返回数值 number 的以 10 为底的对数。如果 number 为 null，则 Number.Log10 返回 null。

Number.Log10(number as nullable number) as nullable number

获取 2 的以 10 为底的对数。

```
Number.Log10(2)
0.3010299956639812
```

● Number.Mod

返回从 number 整除 divisor 所得的余数。

如果 number 或 divisor 为 null，则 Number.Mod 返回 null。

number : 被除数。

divisor : 除数。

Number.Mod(number as nullable number, divisor as nullable number, optional precision as nullable number) as nullable number

计算 5 除以 3 所得的余数。

```
Number.Mod(5, 3)
2
```

● Number.NaN

表示 0 除以 0 的常量值。

● Number.NegativeInfinity

表示 -1 除以 0 的常量值。

● Number.PI

表示 3.1415926535897932 的常量，pi 的值最多取 16 位小数。

● Number.Permutations

使用指定的排列大小 permutationSize 返回可从项数 setSize 生成的排列数。

Number.Permutations(setSize as nullable number, permutationSize as nullable number) as nullable number

计算 3 个一组、从总共 5 项得到的排列数。

```
Number.Permutations(5, 3)
60
```

● Number.PositiveInfinity

表示 1 除以 0 的常量值。

● Number.Power

返回计算 number 的 power 次幂所得的结果。

如果 number 或 power 为 null，则 Number.Power 返回 null。

number：底数。

power：指数。

Number.Power(number as nullable number, power as nullable number) as nullable number

计算 5 的 3 次幂(5 的立方)的值。

```
Number.Power(5, 3)
125
```

● Number.Random

返回介于 0 到 1 之间的随机数。

Number.Random() as number

获取随机数。

Number.Random()
0.919303

● Number.RandomBetween

返回 bottom 和 top 之间的一个随机数。

Number.RandomBetween(bottom as number, top as number) as number

获取 1 和 5 之间的一个随机数。

Number.RandomBetween(1, 5)
2.546797

● Number.Round

将舍入 number 的结果返回最接近的数字。如果 number 为 null， Number.Round 将返回 null。

number 将舍入为最接近的整数，除非指定了可选参数 digits 。如果指定了 digits ，则将 number 舍入为包含 digits 位小数。可选的 roundingMode 参数指定可能要四舍五入的数字之间存在联系时的舍入方向(请参阅 RoundingMode.Type 查看可能的值)。

Number.Round(number as nullable number, optional digits as nullable number, optional roundingMode as nullable number) as nullable number

将 1.234 舍入到最近的整数。

Number.Round(1.234)
1

将 1.56 舍入到最近的整数。

Number.Round(1.56)
2

将 1.2345 舍入为包含两位小数。

Number.Round(1.2345, 2)
1.23

将 1.2345 舍入为包含三位小数(向上舍入)。

Number.Round(1.2345, 3, RoundingMode.Up)
1.235

将 1.2345 舍入为包含三位小数(向下舍入)。

```
Number.Round(1.2345, 3, RoundingMode.Down)
1.234
```

● Number.RoundAwayFromZero

基于数字的符号返回舍入 number 的结果。此函数将向上舍入正数、向下舍入负数。

如果指定了 digits ，则将 number 舍入为包含 digits 位小数。

```
Number.RoundAwayFromZero( number as nullable number, optional digits as nullable number) as nullable number
```

向远离零的方向舍入数 -1.2。

```
Number.RoundAwayFromZero(-1.2)
-2
```

向远离零的方向舍入数 1.2。

```
Number.RoundAwayFromZero(1.2)
2
```

将数字 -1.234 向远离零的方向舍入为包含两位小数。

```
Number.RoundAwayFromZero(-1.234, 2)
-1.24
```

● Number.RoundDown

返回将 number 向下舍入到上一个最大整数的结果。如果 number 为 null ，Number.RoundDown 将返回 null。

如果指定了 digits ，则将 number 舍入为包含 digits 位小数。

```
Number.RoundDown( number as nullable number, optional digits as nullable number) as nullable number
```

将 1.234 向下舍入到整数。

```
Number.RoundDown(1.234)
1
```

将 1.999 向下舍入到整数。

Number.RoundDown(1.999)
1

将 1.999 向下舍入为包含两位小数。

Number.RoundDown(1.999, 2)
1.99

- ## Number.RoundTowardZero

基于数字的符号返回舍入 number 的结果。此函数将向下舍入正数、向上舍入负数。

如果指定了 digits ，则将 number 舍入为包含 digits 位小数。

Number.RoundTowardZero(number as nullable number, optional digits as nullable number) as nullable number

- ## Number.RoundUp

返回将 number 向下舍入到上一个最大整数的结果。如果 number 为 null，

Number.RoundDown 将返回 null。

如果指定了 digits ，则将 number 舍入为包含 digits 位小数。

Number.RoundUp(number as nullable number, optional digits as nullable number) as nullable number

将 1.234 向上舍入到整数。

Number.RoundUp(1.234)
2

将 1.999 向上舍入到整数。

Number.RoundUp(1.999)
2

将 1.234 向上舍入为包含两位小数。

Number.RoundUp(1.234, 2)
1.24

- ## Number.Sign

如果 number 为正数，则返回 1；如果它为负数，则返回 -1；如果它为零则返回 0。

如果 number 为 null，则 Number.Sign 返回 null。

```
Number.Sign( number  as nullable number) as nullable number
```

```
    确定 182 的符号。

    Number.Sign(182)
    1
    ----------------

    确定 -182 的符号。

    Number.Sign(-182)
    -1
    ----------------

    确定 0 的符号。

    Number.Sign(0)
    0
```

Number.Sin

返回 number 的正弦。

```
Number.Sin( number  as nullable number) as nullable number
```

```
    计算角 0 的正弦。

    Number.Sin(0)
    0
```

Number.Sinh

返回 number 的双曲正弦。

```
Number.Sinh( number  as nullable number) as nullable number
```

Number.Sqrt

返回 number 的平方根。

如果 number 为 null，则 Number.Sqrt 返回 null。如果它是负值，则返回 Number.NaN (不是数字)。

```
Number.Sqrt( number  as nullable number) as nullable number
```

```
    计算 625 的平方根。

    Number.Sqrt(625)
    25
```

计算 85 的平方根。

Number.Sqrt(85)
9.2195444572928871

● Number.Tan

返回 number 的正切。

Number.Tan(number as nullable number) as nullable number

计算角 1 的正切。

Number.Tan(1)
1.5574077246549023

● Number.Tanh

返回 number 的双曲正切。

Number.Tanh(number as nullable number) as nullable number

● Number.ToText

根据 format 所指定的格式将数值 number 格式化为文本值。格式为单个字符代码(可选择后跟数字精度说明符)

可以将以下参数代码用于 format 。

"D" 或 "d": (十进制)将结果格式化为整数位。精度说明符控制输出中的位数。

"E" 或 "e": (指数 [科学])指数记数法。精度说明符控制最大小数位数(默认值为 6)。

"F" 或 "f": (固定点)整数和小数位。

"G" 或 "g": (常规)固定点或科学表示法的最简洁形式。

"N" 或 "n": (数字)带组分隔符和小数分隔符的整数和小数位。

"P" 或 "p": (百分比)乘以 100 并显示百分号的数字。

"R" 或 "r": (往返)可往返转换同一数字的文本值。忽略精度说明符。

"X" 或 "x": (十六进制)十六进制文本值。

Number.ToText(number as nullable number, optional format as nullable text, optional culture as nullable text) as nullable text

将数字格式化为不指定格式的文本。

```
Number.ToText(4)
"4"
----------------
```

将数字格式化为指数格式的文本。

```
Number.ToText(4, "e")
"4.000000e+000"
----------------
```

将数字格式化为带有限精确度的十进制格式的文本。

```
Number.ToText(-0.1234, "P1")
"-12.3 %"
```

● Number.Type

表示所有数字的类型。

● OData.Feed

从 URI serviceUri 标头 headers 返回由 OData 服务提供的 OData 源表。可以指定用于指定是使用并发连接还是使用可选记录参数的布尔值 options 以控制以下选项:

Query : 以编程方式向 URL 添加查询参数而无需担心转义。

Headers : 指定此值作为记录以向 HTTP 请求提供额外标头。

ExcludedFromCacheKey : 指定此值作为列表以将这些 HTTP 标头键从缓存数据计算部分排除。

ApiKeyName : 如果目标站点对 API 密钥有一些概念，则此参数可用于指定必须在该 URL 中使用的密钥参数的名称(而不是值)。在凭据中提供实际密钥值。

Timeout : 指定此值作为持续时间将更改 HTTP 请求的超时时间。默认值为 600 秒。

EnableBatch：一个逻辑值(true/false)，用于设置超出 MaxUriLength 时是否允许生成 OData $batch 请求(默认值为 false)。

MaxUriLength：一个数字，表示允许被发送到 OData 服务的 URI 的最大长度。如果 URI 超出了最大长度且 EnableBatch 为 true，则将向 OData $batch 终结点发送请求，否则会失败(默认值为 2048)。

Concurrent：一个逻辑值(true/false)，设置为 true 时，将并发执行对服务的请求。设置为 false 时，将按顺序执行请求。未指定时，将由服务的 AsynchronousRequestsSupported 注释确定该值。如果该服务未指定是否支持 AsynchronousRequestsSupported，将按顺序执行请求。

OData.Feed(serviceUri as text, optional headers as nullable record, optional options as any) as any

- **Occurrence.All**

返回找到的值出现的所有位置的列表。

- **Occurrence.First**

返回找到的值第一次出现的位置。

- **Occurrence.Last**

返回找到的值最后一次出现的位置。

- **Occurrence.Optional**

要求该项在输入中不出现或出现一次。

- **Occurrence.Repeating**

要求该项在输入中不出现或出现多次。

- **Occurrence.Required**

要求该项在输入中出现一次。

● Occurrence.Type

指定序列中元素的出现次数。

● Odbc.DataSource

从连接字符串 connectionString 指定的 ODBC 数据源返回包含 SQL 表和视图的表。

connectionString 可以是文本，也可以是属性值对记录。属性值可以是文本，也可以是数字。

可以提供可选的记录参数 options 来指定附加属性。记录可以包含以下字段:

CreateNavigationProperties :逻辑(true/false)，用于设置是否在返回的值上生成导航属性(默认值为 true)。

HierarchicalNavigation :逻辑(true/false)，用于设置是否查看按架构名称和目录名称(若支持)分组的表(默认值为 false)。

Odbc.DataSource(connectionString as any, optional options as nullable record) as table

● Odbc.Query

返回使用 ODBC 运行带有连接字符串 connectionString 的 query 的结果。

connectionString 可以是文本或属性值对的记录。属性值可以是文本或数字。

Odbc.Query(connectionString as any, query as text) as table

● OleDb.DataSource

返回 SQL 表的表并从连接字符串 connectionString 指定的 OLE DB 数据源进行查看。

connectionString 可以是文本或属性值对的记录。属性值可以是文本或数字。可以提供一个可选的记录参数 options 来指定其他属性。记录可以包含以下字段:

CreateNavigationProperties : 一个逻辑值(true/false)，用于在返回的值上设置是否生成导航属性(默认值为 true)。

Query : 在数据源上作为本机查询运行的文本。

HierarchicalNavigation ：一个逻辑值(true/false)，用于设置是否在支持的情况下查看按其架构名称和目录名称分组的表，默认为 true)。

例如，将记录参数指定为 [option1 = value1, option2 = value2...] 或 [Query = "select ..."]。

```
OleDb.DataSource( connectionString as any, optional options as nullable record) as table
```

- **OleDb.Query**

返回使用 OLE DB 运行带有连接字符串 connectionString 的 query 的结果。

connectionString 可以是文本或属性值对的记录。属性值可以是文本或数字。

```
OleDb.Query( connectionString as any, query as text) as table
```

- **Oracle.Database**

从服务器 server 上的 Oracle 数据库返回包含 SQL 表和视图的表。可以视需要使用服务器指定端口，并用冒号分隔。可以指定可选的记录参数 options 来控制以下选项:

CreateNavigationProperties :逻辑(true/false)，用于设置是否在返回的值上生成导航属性(默认值为 true)。

NavigationPropertyNameGenerator :函数，用于创建导航属性的名称。

Query :文本，用于转换为在服务器上运行的 SQL 查询。你可以指定多个查询，但系统只返回第一个结果。

CommandTimeout :持续时间，用于控制服务器端查询在取消前运行的时间。默认值为十分钟。

HierarchicalNavigation :逻辑(true/false)，用于设置是否查看按架构名称分组的表(默认值为false)。

例如，可以将记录参数指定为 [option1 = value1, option2 = value2...] 或 [Query = "select ..."]。

```
Oracle.Database( server as text, optional options as nullable record) as table
```

- **Order.Ascending**

以升序排序列表的函数类型。

- **Order.Descending**

以降序排序列表的函数类型。

- **Order.Type**

指定排序方向。

- **Percentage.From**

返回给定 value 的 百分比 值。如果给定的 value 为 null ，则 Percentage.From 返回 null 。如果给定的 value 是带有尾随百分比符号的 文本 ，则返回转换的小数。否则，请查看将其转换为 数 值的 Number.From 。

```
Percentage.From( value  as any, optional  culture  as nullable text) as nullable number
```

获取 "12.3%" 的百分比值。

```
Percentage.From("12.3%")
0.123
```

- **Percentage.Type**

表示百分比值的类型。

- **PostgreSQL.Database**

返回包含(在名为 database 的数据库实例中)服务器 server 上 PostgreSQL 数据库中可用的 SQL 表和视图的表。可以视需要使用服务器指定端口，并用冒号分隔。可以指定可选的记录参数 options 来控制以下选项:

CreateNavigationProperties :逻辑(true/false)，用于设置是否在返回的值上生成导航属性(默认值为 true)。

NavigationPropertyNameGenerator :函数，用于创建导航属性的名称。

Query :文本，用于转换为在服务器上运行的 SQL 查询。你可以指定多个查询，但系统只返回第一个结果。

CommandTimeout :持续时间，用于控制服务器端查询在取消前运行的时间。默认值为十分钟。

HierarchicalNavigation :逻辑(true/false)，用于设置是否查看按架构名称分组的表(默认值为false)。

例如，可以将记录参数指定为 [option1 = value1, option2 = value2...] 或 [Query = "select ..."]。

PostgreSQL.Database(server as text, database as text, optional options as nullable record) as table

- **Precision.Decimal**

内置算术运算符的可选参数，以指定小数精度。

- **Precision.Double**

内置算术运算符的可选参数，以指定双精度。

- **Precision.Type**

指定比较的精度。

- **QuoteStyle.Csv**

引号字符指示被引用的字符串的开头。 嵌套的引号由两个引号字符指示。

- **QuoteStyle.None**

引号字符没有意义。

- **QuoteStyle.Type**

指定引号样式。

None : (默认值)无需任何加引号行为。

Csv：如何加引号按 CSV 的要求进行。使用一个双引号字符来界定这些区域，使用一对双引号来表示此区域中的单个双引号字符。

- **RData.FromBinary**

从 RData 文件返回数据帧记录。

RData.FromBinary(stream as binary) as any

- **Record.AddField**

给定字段 fieldName 的名称和值 value ，将字段添加到记录 record 。

Record.AddField(record as record, fieldName as text, value as any, optional delayed as nullable logical) as record

将字段地址添加到记录。

Record.AddField([CustomerID = 1, Name = "Bob", Phone = "123-4567"], "Address", "123 Main St.")
[CustomerID=1, Name= "Bob", Phone="123-4567", Address="123 Main St."]

- **Record.Combine**

组合给定 records 中的记录。如果 records 包含非记录值，将返回错误。

Record.Combine(records as list) as record

从记录创建组合记录。

Record.Combine({ [CustomerID =1, Name ="Bob"] , [Phone = "123-4567"]})
[CustomerID=1, Name="Bob", Phone="123-4567"]

- **Record.Field**

返回 record 中指定 field 的值。如果未找到该字段，将引发异常。

Record.Field(record as record, field as text) as any

在记录中查找字段 "CustomerID" 的值。

Record.Field([CustomerID = 1, Name = "Bob", Phone = "123-4567"], "CustomerID")
1

- **Record.FieldCount**

返回记录 record 中的字段数。

Record.FieldCount(record as record) as number

查找记录中的字段数。

```
Record.FieldCount([CustomerID = 1, Name = "Bob"])
2
```

● Record.FieldNames

将记录 record 中的字段名称作为文本返回。

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

```
查找记录中字段的名称。

Record.FieldNames([OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0])
{"OrderID","CustomerID", "Item", "Price"}
```

● Record.FieldOrDefault

返回记录 record 中指定字段 field 的值。如果未找到该字段，则返回可选的 defaultValue 。

```
Record.FieldOrDefault( record as record, field as text, optional defaultValue as any) as any
```

```
在记录中查找字段 "Phone" 的值，如果它不存在，则返回 null。

Record.FieldOrDefault([CustomerID =1, Name="Bob"], "Phone")
null
----------

在记录中查找字段 "Phone" 的值，如果它不存在，则返回默认值。

Record.FieldOrDefault([CustomerID =1, Name="Bob"], "Phone", "123-4567")
"123-4567"
```

● Record.FieldValues

返回记录 record 中的字段值列表。

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

```
在记录中查找字段值。

Record.FieldValues([CustomerID = 1, Name = "Bob", Phone = "123-4567"])
{1, "Bob", "123-4567"}
```

● Record.FromList

根据给定的一个字段值 list 和一组字段，返回一个记录。 可以通过文本值列表或记录类型指定 fields 。 如果字段不是唯一的，将引发错误。

```
Record.FromList( list as list, fields as any) as record
```

从一个字段值列表和字段名称列表生成一个记录。

```
Record.FromList({1, "Bob", "123-4567"}, {"CustomerID", "Name", "Phone"})
[CustomerID = 1, Name = "Bob", Phone = "123-4567"]
```

从一个字段值列表和记录类型生成一个记录。

```
Record.FromList({1, "Bob", "123-4567"}, type [CustomerID = number, Name = text, Phone = number])
[CustomerID = 1, Name = "Bob", Phone = "123-4567"]
```

● Record.FromTable

从包含字段名称和值名称 {[Name = name, Value = value]} 的记录 table 的表返回记录。

如果字段名称不是唯一的，将引发异常。

```
Record.FromTable( table as table) as record
```

从 Table.FromRecords({[Name = "CustomerID", Value = 1], [Name = "Name", Value = "Bob"], [Name = "Phone", Value = "123-4567"]})格式的表创建记录。

```
Record.FromTable(Table.FromRecords({[Name = "CustomerID", Value = 1], [Name = "Name", Value = "Bob"], [Name = "Phone", Value = "123-4567"]}))
[CustomerID = 1, Name = "Bob", Phone = "123-4567"]
```

● Record.HasFields

通过返回逻辑值(true 或 false)，指示记录 record 是否具有 fields 中指定的字段。

可以使用列表指定多个字段值。

```
Record.HasFields( record as record, fields as any) as logical
```

检查记录是否包含字段 "CustomerID"。

```
Record.HasFields([CustomerID = 1, Name = "Bob", Phone = "123-4567"],"CustomerID")
true
```

检查记录是否包含字段 "CustomerID" 和 "Address"。

```
Record.HasFields([CustomerID = 1, Name = "Bob", Phone = "123-4567"],{"CustomerID", "Address"})
false
```

● Record.RemoveFields

返回一个记录，该记录从输入 record 中删除列表 fields 中指定的所有字段。如果指定的字段不存在，将引发异常。

Record.RemoveFields(record as record, fields as any, optional missingField as nullable number) as record

从记录中删除字段 "Price"。

Record.RemoveFields([CustomerID=1, Item = "Fishing rod", Price=18.00], "Price")
[CustomerID=1, Item="Fishing rod"]

从记录中删除字段 "Price" 和 "Item"。

Record.RemoveFields([CustomerID=1, Item = "Fishing rod", Price=18.00], {"Price", "Item"})
[CustomerID=1]

● Record.RenameFields

将输入 record 中的字段重命名为列表 renames 中指定的新字段名称后，返回一个记录。对于多个重命名，可以使用嵌套的列表({ {old1, new1}, {old2, new2} })。

Record.RenameFields(record as record, renames as list, optional missingField as nullable number) as record

从记录将字段 "UnitPrice" 重命名为 "Price"。

Record.RenameFields([OrderID = 1, CustomerID = 1, Item = "Fishing rod", UnitPrice = 100.0], {"UnitPrice","Price"})
[OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0]

从记录将字段 "UnitPrice" 重命名为 "Price"，将字段 "OrderNum" 重命名为 "OrderID"。

Record.RenameFields([OrderNum = 1, CustomerID = 1, Item = "Fishing rod", UnitPrice = 100.0], {{"UnitPrice", "Price"}, {"OrderNum", "OrderID"}})
[OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0]

● Record.ReorderFields

按列表 fieldOrder 中指定的字段顺序将 record 中的字段重新排序后，返回一个记录。字段值保持不变， fieldOrder 中未列出的字段仍留在原始位置。

Record.ReorderFields(record as record, fieldOrder as list, optional missingField as nullable number) as record

将记录中的部分字段重新排序。

Record.ReorderFields([CustomerID= 1, OrderID = 1, Item = "Fishing rod", Price = 100.0], {"OrderID", "CustomerID"})
[OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0]

● Record.SelectFields

从输入 record 返回一个记录，该记录仅包含在列表 fields 中指定的字段。

Record.SelectFields(record as record, fields as any, optional missingField as nullable number) as record

在记录中选择字段 "Item" 和 "Price"。

Record.SelectFields([OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0] , {"Item", "Price"})
[Item = "Fishing rod", Price = 100]

● Record.ToList

返回包含输入 record 中的字段值的值列表。

Record.ToList(record as record) as list

从记录提取字段值。

Record.ToList([A = 1, B = 2, C = 3])
{1, 2, 3}

● Record.ToTable

返回一个表，它包含 Name 和 Value 列以及对应于 record 中每个字段的行。

Record.ToTable(record as record) as table

从记录返回表。

Record.ToTable([OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0])
Table.FromRecords({[Name="OrderID", Value=1],[Name="CustomerID", Value=1],[Name="Item", Value="Fishing rod"],[Name="Price", Value=100]})

● Record.TransformFields

将列表 transformOperations 中指定的转换应用到 record 后返回一个记录。

一次可以转换一个或多个字段。

在转换单个字段时，要求 transformOperations 是包含两项的列表。 transformOperations

中的第一项指定字段名称， transformOperations 中的第二项指定要用于转换的函数。例如 {"Quantity", Number.FromText}

在转换多个字段时，要求 transformOperations 是列表的列表，其中每个里面的列表是一对字段名称和转换操作。例如 {{"Quantity",Number.FromText},{"UnitPrice", Number.FromText}}

```
Record.TransformFields( record as record, transformOperations as list, optional missingField as nullable number) as record

    将 "Price" 字段转换为数字。

    Record.TransformFields([OrderID = 1, CustomerID= 1, Item = "Fishing rod", Price = "100.0"], {"Price", Number.FromText})
    [OrderID = 1, CustomerID= 1, Item = "Fishing rod", Price = 100]
    ----------

    将 "OrderID" 和 "Price" 字段转换为数字。

    Record.TransformFields(
    [OrderID ="1", CustomerID= 1, Item = "Fishing rod", Price = "100.0"],
    {{"OrderID", Number.FromText}, {"Price",Number.FromText}})
    [OrderID = 1, CustomerID= 1, Item = "Fishing rod", Price = 100]
```

● Record.Type

表示所有记录的类型。

● Replacer.ReplaceText

使用 new 文本替换原始 text 中的 old 文本。可以在 List.ReplaceValue 和 Table.ReplaceValue 中使用此替换器函数。

```
Replacer.ReplaceText( text as nullable text, old as text, new as text) as nullable text

    在字符串 "hEllo world" 中使用 "He" 替换文本 "hE"。

    Replacer.ReplaceText("hEllo world","hE","He")
    "Hello world"
```

● Replacer.ReplaceValue

使用 new 值替换原始 value 中的 old 值。可以在 List.ReplaceValue 和 Table.ReplaceValue 中使用此替换器函数。

```
Replacer.ReplaceValue( value as any, old as any, new as any) as any

    使用值 10 替换值 11。

    Replacer.ReplaceValue(11, 11, 10)
    10
```

Resource.Access

Resource.Access

```
Resource.Access( resource  as any, optional  nativeQuery  as nullable text) as any
```

RoundingMode.AwayFromZero

RoundingMode.AwayFromZero

RoundingMode.Down

RoundingMode.Down

RoundingMode.ToEven

RoundingMode.ToEven

RoundingMode.TowardZero

RoundingMode.TowardZero

RoundingMode.Type

指定当要舍入到的可能值之间存在等同值时的舍入方向。

RoundingMode.Up

RoundingMode.Up

RowExpression.Column

返回表示对行表达式内的列 columnName 的访问权限的 AST。

```
RowExpression.Column( columnName  as text) as record
```

```
创建表示对列 "CustomerName" 的访问权限的 AST。

RowExpression.Column("CustomerName")
[
Kind = "FieldAccess",
Expression = RowExpression.Row,
MemberName = "CustomerName"
]
```

RowExpression.From

返回 function 主体的 AST，规范化为行表达式:

函数必须为 1 个参数的 lambda。

对函数参数的所有引用已替换为 RowExpression.Row 。

对列的所有引用已替换为 RowExpression.Column(columnName) 。

AST 将简化为仅包含以下类型的节点:

Constant
Invocation
Unary
Binary
If
FieldAccess

如果无法返回 function 的主体的行表达式 AST，则会出现错误。

```
RowExpression.From( function  as function) as record

    返回函数 each [CustomerID] = "ALFKI" 主体的 AST

    RowExpression.From(each [CustomerName] = "ALFKI")
    [
    Kind = "Binary",
    Operator = "Equals",
    Left = RowExpression.Column("CustomerName"),
    Right =
    [
    Kind = "Constant",
    Value = "ALFKI"
    ]
    ]
```

- **RowExpression.Row**

表示行表达式中的行的 AST 节点。

- **Salesforce.Data**

返回凭据中提供的 Salesforce 帐户上的对象。该帐户将通过提供的环境 loginUrl 进行连接。如果没有提供任何环境，则该帐户将连接到生产环境(https://login.salesforce.com)。可提供可选记录参数 options 来指定附加属性。该记录可以包含以下字段:

CreateNavigationProperties : 一个逻辑值(true/false)，可用于设置是否对返回值生成导航属

性(默认为 false)。

Salesforce.Data(optional loginUrl as any, optional options as nullable record) as table

- **Salesforce.Reports**

返回凭据中提供的 Salesforce 帐户上的报告。该帐户将通过提供的环境 loginUrl 进行连接。如果没有提供任何环境，则该帐户将连接到生产环境(https://login.salesforce.com)。

Salesforce.Reports(optional loginUrl as nullable text, optional options as nullable record) as table

- **SapHana.Database**

从 SAP HANA 数据库 server 返回多维包表。可以指定可选的记录参数 options 来控制以下选项:

Query ：用于检索数据的本机 SQL 查询。

SapHana.Database(server as text, optional options as nullable record) as table

- **SapHanaRangeOperator.Equals**

SAP HANA 输入参数的“等于”范围运算符。

- **SapHanaRangeOperator.GreaterThan**

SAP HANA 输入参数的“大于”范围运算符。

- **SapHanaRangeOperator.GreaterThanOrEquals**

SAP HANA 输入参数的“大于或等于”范围运算符。

- **SapHanaRangeOperator.LessThan**

SAP HANA 输入参数的“小于”范围运算符。

- **SapHanaRangeOperator.LessThanOrEquals**

SAP HANA 输入参数的“小于或等于”范围运算符。

- **SapHanaRangeOperator.NotEquals**

SAP HANA 输入参数的“不等于”范围运算符。

SapHanaRangeOperator.Type

SAP HANA 范围输入参数的范围运算符。

SharePoint.Contents

返回包含在指定 SharePoint 站点 url 上找到的每个文件夹和文档的行的表。每行都包含该文件夹或文件的属性以及指向其内容的链接。可以指定 options 以控制以下选项:

ApiVersion : 数字(14 或 15)或文本 "Auto" ，指定要用于此站点的 SharePoint API 版本。未指定时，使用 API 版本 14。指定 Auto 时，如果可能将自动发现服务器版本，否则版本默认为 14。非英文的 SharePoint 网站至少需要版本 15。

SharePoint.Contents(url as text, optional options as nullable record) as table

SharePoint.Files

返回包含在指定 SharePoint 站点 url 上找到的每个文档和子文件夹的行的表。每行都包含该文件夹或文件的属性以及指向其内容的链接。可以指定 options 以控制以下选项:

ApiVersion : 数字(14 或 15)或文本 "Auto" ，指定要用于此站点的 SharePoint API 版本。未指定时，使用 API 版本 14。指定 Auto 时，如果可能将自动发现服务器版本，否则版本默认为 14。非英文的 SharePoint 网站至少需要版本 15。

SharePoint.Files(url as text, optional options as nullable record) as table

SharePoint.Tables

返回包含在指定 SharePoint 列表 url 上找到的每个列表项的行的表。每行都包含该列表的属性。可以指定 options 以控制以下选项:

ApiVersion : 数字(14 或 15)或文本 "Auto" ，指定要用于此站点的 SharePoint API 版本。未指定时，使用 API 版本 14。指定 Auto 时，如果可能将自动发现服务器版本，否则版本默认为 14。非英文的 SharePoint 网站至少需要版本 15。

SharePoint.Tables(url as text, optional options as nullable record) as table

Single.From

从给定的 value 返回 Single number 值。如果给定的 value 为 null ， Single.From 将返回 null 。如果给定的 value 为 Single 范围内的 number ，将返回 value ，否则返回错误。如果给定的 value 为其他类型，请参阅 Number.FromText ，将其转换为 number 值，此后有关将 number 值转换为 Single number 值的上一语句适用。

```
Single.From( value  as any, optional  culture  as nullable text) as nullable number
```

获取 "1.5" 的单精度 number 值。

```
Single.From("1.5")
```

1.5

Single.Type

表示单精度浮点数的类型。

Soda.Feed

从位于指定 URL url (根据 SODA 2.0 API 进行格式化)的内容中返回一个表。URL 必须指向有效的符合 SODA 的源(以 .csv 扩展名结尾)。

```
Soda.Feed( url  as text) as table
```

Splitter.SplitByNothing

返回不拆分且将其自变量作为单元素列表返回的函数。

```
Splitter.SplitByNothing() as function
```

Splitter.SplitTextByAnyDelimiter

返回一个函数，它在任意指定的分隔符处将文本拆分为文本列表。

```
Splitter.SplitTextByAnyDelimiter( delimiters  as list, optional  quoteStyle  as nullable number, optional startAtEnd  as nullable logical) as function
```

Splitter.SplitTextByDelimiter

返回一个函数，它根据指定的分隔符将文本拆分为文本列表。

```
Splitter.SplitTextByDelimiter( delimiter  as text, optional  quoteStyle  as nullable number) as function
```

Splitter.SplitTextByEachDelimiter

返回一个函数，它依次在每个指定的分隔符处将文本拆分为文本列表。

Splitter.SplitTextByEachDelimiter(delimiters as list, optional quoteStyle as nullable number, optional startAtEnd as nullable logical) as function

- **Splitter.SplitTextByLengths**

返回一个函数，它按每个指定的长度将文本拆分为文本列表。

Splitter.SplitTextByLengths(lengths as list, optional startAtEnd as nullable logical) as function

- **Splitter.SplitTextByPositions**

返回一个函数，它在每个指定的位置将文本拆分为文本列表。

Splitter.SplitTextByPositions(positions as list, optional startAtEnd as nullable logical) as function

- **Splitter.SplitTextByRanges**

返回一个函数，它根据指定的偏移量和长度将文本拆分为文本列表。

Splitter.SplitTextByRanges(ranges as list, optional startAtEnd as nullable logical) as function

- **Splitter.SplitTextByRepeatedLengths**

返回一个函数，它在指定的长度后反复将文本拆分为文本列表。

Splitter.SplitTextByRepeatedLengths(length as number, optional startAtEnd as nullable logical) as function

- **Splitter.SplitTextByWhitespace**

返回一个函数，它在空白处将文本拆分为文本列表。

Splitter.SplitTextByWhitespace(optional quoteStyle as nullable number) as function

- **Sql.Database**

从服务器 server 上的 SQL Server 数据库 database 返回包含 SQL 表、视图和存储函数的表。可选择通过服务器指定端口，并用冒号或逗号分隔。可以指定可选的记录参数 options 来控制以下选项:

MaxDegreeOfParallelism：一个数字，用于设置生成的 SQL 查询中 "maxdop" 查询子句的值。

CreateNavigationProperties：一个逻辑值(true/false)，用于设置是否在返回的值上生成导航属性(默认值为 true)。

NavigationPropertyNameGenerator：一个函数，用于创建导航属性的名称。

Query：文本，用于转换为在服务器上运行的 SQL 查询。你可以指定多个查询，但系统只返回第一个结果。

CommandTimeout：持续时间，用于控制服务器端查询在取消前运行的时间。默认值为十分钟。

HierarchicalNavigation：一个逻辑值(true/false)，用于设置是否查看按架构名称分组的表(默认值为 false)。

MultiSubnetFailover ：一个逻辑值(true/false)，用于设置连接字符串中的"MultiSubnetFailover" 属性的值(默认值为 false)。

例如，将记录参数指定为 [option1 = value1, option2 = value2...] 或 [Query = "select ..."]。

Sql.Database(server as text, database as text, optional options as nullable record) as table

- **Sql.Databases**

返回指定 SQL Server server 上的数据库表。可以指定可选的记录参数 options 来控制以下选项:

MaxDegreeOfParallelism：一个数字，用于设置生成的 SQL 查询中 "maxdop" 查询子句的值。

CreateNavigationProperties：一个逻辑值(true/false)，用于设置是否在返回的值上生成导航属性(默认值为 true)。

NavigationPropertyNameGenerator：一个函数，用于创建导航属性的名称。

CommandTimeout : 持续时间，用于控制服务器端查询在取消前运行的时间。默认值为十分钟。

HierarchicalNavigation : 一个逻辑值(true/false)，用于设置是否查看按架构名称分组的表(默认值为 false)。

MultiSubnetFailover : 一个逻辑值(true/false)，用于设置连接字符串中"MultiSubnetFailover" 属性的值(默认值为 false)。

例如，将记录参数指定为 [option1 = value1, option2 = value2...]。

不支持设置 SQL 查询以在服务器上运行。不应使用 Sql.Database 来代替运行 SQL 查询。

Sql.Databases(server as text, optional options as nullable record) as table

● SqlExpression.SchemaFrom

SqlExpression.SchemaFrom

SqlExpression.SchemaFrom(schema as any) as any

● SqlExpression.ToExpression

SqlExpression.ToExpression

SqlExpression.ToExpression(sql as text, types as type) as text

● Sybase.Database

返回包含(在名为 database 的数据库实例中)服务器 server 上 Sybase 数据库中可用的 SQL 表和视图的表。可以视需要使用服务器指定端口，并用冒号分隔。可以指定可选的记录参数 options 来控制以下选项:

CreateNavigationProperties :逻辑(true/false)，用于设置是否在返回的值上生成导航属性(默认值为 true)。

NavigationPropertyNameGenerator :函数，用于创建导航属性的名称。

Query :文本，用于转换为在服务器上运行的 SQL 查询。你可以指定多个查询，但系统只返回第一个结果。

CommandTimeout :持续时间，用于控制服务器端查询在取消前运行的时间。默认值为十分钟。

HierarchicalNavigation :逻辑(true/false)，用于设置是否查看按架构名称分组的表(默认值为false)。

例如，可以将记录参数指定为 [option1 = value1, option2 = value2...] 或 [Query = "select ..."]。

Sybase.Database(server as text, database as text, optional options as nullable record) as table

● Table.AddColumn

将名为 newColumnName 的列添加到表 table 。使用指定的选择函数 columnGenerator (它将每行作为输入)来计算列的值。

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

将名为 "TotalPrice" 的列添加到表，其中每个值是列 [Price] 和列 [Shipping] 的和。

```
Table.AddColumn(Table.FromRecords({[OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0, Shipping = 10.00], [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0, Shipping = 15.00], [OrderID = 3, CustomerID = 2, Item = "Fishing net", Price = 25.0, Shipping = 10.00]}), "TotalPrice", each [Price] + [Shipping])
Table.FromRecords({[OrderID=1,CustomerID=1,Item="Fishing rod", Price=100,Shipping=10,TotalPrice=110],[OrderID=2,CustomerID=1,Item="1 lb. worms", Price=5,Shipping=15,TotalPrice=20],[OrderID=3,CustomerID=2,Item="Fishing net", Price=25,Shipping=10,TotalPrice=35]})
```

● Table.AddIndexColumn

使用显式位置值将名为 newColumnName 的列追加到 table 。

可选值 initialValue 为初始索引值。可选值 increment 指定每个索引值的增量值。

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

将名为 "Index" 的索引列添加到表。

Table.AddIndexColumn(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), "Index")

Table.FromRecords({[CustomerID=1,Name="Bob",Phone="123-4567", Index=0],[CustomerID=2,Name="Jim",Phone="987-6543", Index=1],[CustomerID=3,Name="Paul",Phone="543-7890", Index=2],[CustomerID=4,Name="Ringo",Phone="232-1550", Index=3]})

从值 10 开始、按 5 递增将名为 "index" 的索引列添加到表。

Table.AddIndexColumn(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), "Index", 10, 5)

Table.FromRecords({[CustomerID=1,Name="Bob",Phone="123-4567", Index=10],[CustomerID=2,Name="Jim",Phone="987-6543",Index=15],[CustomerID=3,Name="Paul",Phone="543-7890",Index=20...

● Table.AddJoinColumn

根据由 key1 (针对 table1)和 key2 (针对 table2)选择的键列的值是否相等，联接 table1 的行与 table2 的行。将结果输入到名为 newColumnName 的列。

此函数的行为类似于具有 JoinKind 的 LeftOuter 的 Table.Join，除了联接结果以嵌套而非平展方式存在之外。

Table.AddJoinColumn(table1 as table, key1 as any, table2 as any, key2 as any, newColumnName as text) as table

从已联接 [saleID] 的表({[saleID = 1, price = 20], [saleID = 2, price = 10]})将名为 "price/stock" 的联接列添加到({[saleID = 1, item = "Shirt"], [saleID = 2, item = "Hat"]})。

Table.AddJoinColumn(Table.FromRecords({[saleID = 1, item = "Shirt"], [saleID = 2, item = "Hat"]}), "saleID", () => Table.FromRecords({[saleID = 1, price = 20, stock = 1234], [saleID = 2, price = 10, stock = 5643]}), "saleID", "price")

Table.FromRecords({ [
saleID = 1,
item = "Shirt",
price =Table.FromRecords({ [
saleID = 1,
price = 20,
stock = 1234

```
]
}, {
"saleID",
"price",
"stock"
})
], [
saleID = 2,
item = "Hat",
price =Table.FromRecords({ [
saleID = 2,
price = 10,
stock = 5643
]
}, {
"saleID",
"price",
"stock"
})
]
}, {
"saleID",
"item"...
```

● Table.AddKey

将一个键添加到 table ，给定的 columns 是定义该键的 table 的列名称的子集，并且 isPrimary 指定该键是否为主键。

Table.AddKey(table as table, columns as list, isPrimary as logical) as table

向包含 {"Id"} 的 {[Id = 1, Name = "Hello There"], [Id = 2, Name = "Good Bye"]} 添加键并使其成为主键。

```
let
tableType = type table [Id = Int32.Type, Name = text],
table = Table.FromRecords({[Id = 1, Name = "Hello There"], [Id = 2, Name = "Good Bye"]}),
resultTable = Table.AddKey(table, {"Id"}, true)
in
resultTable
Table.FromRecords({[Id = 1, Name = "Hello There"], [Id = 2, Name = "Good Bye"]}, {
"Id",
"Name"
})
```

● Table.AggregateTableColumn

将 table [column] 中的表聚合到包含这些表的聚合值的多个列。 aggregations 用于指定包含要聚合的表的列、要应用于表以生成其值的聚合函数以及要创建的聚合列的名称。

Table.AggregateTableColumn(table as table, column as text, aggregations as list) as table

将表 {[t = {[a=1, b=2, c=3], [a=2,b=4,c=6]}, b = 2]} 中的 [t] 的表列聚合为 [t.a]、[t.b] 的最小值和最大值以及 [t.a] 中值计数的总和。

```
Table.AggregateTableColumn(Table.FromRecords({[t = Table.FromRecords({[a=1, b=2, c=3], [a=2,b=4,c=6]}), b = 2]}, type table [t = table [a=number, b=number, c=number], b = number]), "t", {{"a", List.Sum, "sum of t.a"}, {"b", List.Min, "min of t.b"}, {"b", List.Max, "max of t.b"}, {"a", List.Count, "count of t.a"}})
Table.FromRecords({[#"sum of t.a" = 3, #"min of t.b" = 2, #"max of t.b" = 4, #"count of t.a" = 2, b = 2]})
```

● Table.AlternateRows

保留初始偏移量，然后交替选取和跳过下列行。

table： 输入表。

offset： 在开始迭代之前要保留的行数。

skip： 每次迭代中要删除的行数。

take： 每次迭代中要保留的行数。

Table.AlternateRows(table as table, offset as number, skip as number, take as number) as table

从表中创建一个表，从第一行开始，跳过 1 个值，然后保留 1 个值。

```
Table.AlternateRows(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"]}), 1, 1, 1)
Table.FromRecords({[CustomerID=1,Name="Bob",Phone="123-4567"],[CustomerID=3,Name="Paul",Phone="543-7890"]})
```

● Table.Buffer

在内存中缓冲一个表，同时在计算期间使其与外部更改隔离。

Table.Buffer(table as table) as table

● Table.Column

将表 table 中由 column 指定的数据列返回为列表。

Table.Column(table as table, column as text) as list

返回表中 [Name] 列的值。

Table.Column(Table.FromRecords({ [CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), "Name")
{"Bob", "Jim", "Paul", "Ringo"}

● Table.ColumnCount

返回表 table 中的列数。

Table.ColumnCount(table as table) as number

查找表中的列数。

Table.ColumnCount(Table.FromRecords({[CustomerID =1, Name ="Bob", Phone = "123-4567"],[CustomerID =2, Name ="Jim", Phone = "987-6543"],[CustomerID =3, Name ="Paul", Phone = "543-7890"]}))
3

● Table.ColumnNames

返回表 table 中的列名作为文本列表。

Table.ColumnNames(table as table) as list

查找表的列名。

Table.ColumnNames(Table.FromRecords({ [CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"] , [CustomerID = 3, Name = "Paul", Phone = "543-7890"] , [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}))
{"CustomerID", "Name", "Phone"}

● Table.ColumnsOfType

返回带有与 listOfTypes 中指定的类型相匹配的表 table 中的列名的列表。

Table.ColumnsOfType(table as table, listOfTypes as list) as list

返回表中类型 Number.Type 的列名。

Table.ColumnsOfType(Table.FromRecords({[a=1,b="hello"]}, type table[a=Number.Type, b=Text.Type]), {type number})
{"a"}

● Table.Combine

返回作为合并一系列表 tables 的结果的表。这些表必须全部具有相同的行类型结构。

Table.Combine(tables as list) as table

将以下三个表合并在一起。

Table.Combine({Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}), Table.FromRecords({[CustomerID = 2, Name = "Jim", Phone = "987-6543"] }),Table.FromRecords({[CustomerID = 3, Name = "Paul", Phone = "543-7890"]})})

Table.FromRecords({[CustomerID=1,Name="Bob",Phone="123-4567"],[CustomerID=2,Name="Jim",Phone="987-6543"],[CustomerID=3,Name="Paul",Phone="543-7890"]})

● Table.CombineColumns

使用指定的组合程序函数将指定的列组合为一个新列。

Table.CombineColumns(table as table, sourceColumns as list, combiner as function, column as text) as table

● Table.Contains

指示指定的记录 row 是否显示为 table 中的一行。

可以指定一个可选参数 equationCriteria ，以控制表各行之间的比较。

Table.Contains(table as table, row as record, optional equationCriteria as any) as logical

确定表是否包含行。

Table.Contains(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), [Name="Bob"])

true

确定表是否包含行。

Table.Contains(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), [Name="Ted"])

false

确定表是否包含只比较列 [Name] 的行。

Table.Contains(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), [CustomerID=4, Name="Bob"], "Name")

true

● Table.ContainsAll

指示记录列表 rows 中所有指定的记录是否显示为 table 中的各行。

可以指定一个可选参数 equationCriteria ，以控制表各行之间的比较。

Table.ContainsAll(table as table, rows as list, optional equationCriteria as any) as logical

确定表是否包含只比较列 [CustomerID] 的所有行。

```
Table.ContainsAll( Table.FromRecords( { [CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"] , [CustomerID = 3, Name = "Paul", Phone = "543-7890"] , [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), {[CustomerID=1, Name="Bill"],[CustomerID=2, Name="Fred"]}, "CustomerID")
true
----------
```

确定表是否包含所有行。

```
Table.ContainsAll( Table.FromRecords( { [CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"] , [CustomerID = 3, Name = "Paul", Phone = "543-7890"] , [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), {[CustomerID=1, Name="Bill"],[CustomerID=2, Name="Fred"]})
false
```

● Table.ContainsAny

指示记录列表 rows 中任何指定的记录是否显示为 table 中的各行。

可以指定一个可选参数 equationCriteria ，以控制表各行之间的比较。

Table.ContainsAny(table as table, rows as list, optional equationCriteria as any) as logical

确定表 ({[a = 1, b = 2], [a = 3, b = 4]}) 是否包含行 [a = 1, b = 2] 或 [a = 3, b = 5]。

```
Table.ContainsAny(Table.FromRecords({[a = 1, b = 2], [a = 3, b = 4]}), {[a = 1, b = 2], [a = 3, b = 5]})
true
----------
```

确定表 ({[a = 1, b = 2], [a = 3, b = 4]}) 是否包含行 [a = 1, b = 3] 或 [a = 3, b = 5]。

```
Table.ContainsAny(Table.FromRecords({[a = 1, b = 2], [a = 3, b = 4]}), {[a = 1, b = 3], [a = 3, b = 5]})
false
----------
```

确定表 (Table.FromRecords({[a = 1, b = 2], [a = 3, b = 4]})) 是否包含只比较列 [a] 的行 [a = 1, b = 3] 或 [a = 3, b = 5]。

```
Table.ContainsAny(Table.FromRecords({[a = 1, b = 2], [a = 3, b = 4]}), {[a = 1, b = 3], [a = 3, b = 5]}, "a")
```

true

● Table.DemoteHeaders

将列标题(也即列名)降级到第一行值。默认列名为 "Column1"、"Column2" 等。

Table.DemoteHeaders(table as table) as table

降级表中的第一行值。

Table.DemoteHeaders(Table.FromRecords({[CustomerID=1, Name="Bob", Phone="123-4567"],[CustomerID=2, Name="Jim", Phone="987-6543"]}))
Table.FromRecords({[Column1="CustomerID", Column2="Name", Column3="Phone"],[Column1=1, Column2="Bob", Column3="123-4567"],[Column1=2, Column2="Jim", Column3="987-6543"]})

● Table.Distinct

从表 table 中删除重复的行。

可选参数 equationCriteria 指定对表中的哪些列进行测试以确定是否具有重复项。如果未指定 equationCriteria ，则测试所有列。

Table.Distinct(table as table, optional equationCriteria as any) as table

从表中删除重复的行。

Table.Distinct(Table.FromRecords({[a = "A", b = "a"], [a = "B", b = "b"], [a = "A", b = "a"]}))
Table.FromRecords({[a = "A", b = "a"],
[a = "B", b = "b"]}, {
"a",
"b"
})

从表 ({[a = "A", b = "a"], [a = "B", b = "a"], [a = "A", b = "b"]}) 的列 [b] 中删除重复的行。

Table.Distinct(Table.FromRecords({[a = "A", b = "a"], [a = "B", b = "a"], [a = "A", b = "b"]}), "b")
Table.FromRecords({[a = "A", b = "a"],
[a = "A", b = "b"]}, {
"a",
"b"
})

● Table.DuplicateColumn

将名为 columnName 的列复制到表 table 。列 newColumnName 的值和类型从列 columnName 复制。

Table.DuplicateColumn(table as table, columnName as text, newColumnName as text, optional columnType as nullable type) as table

将列 "a" 复制到表 ({[a = 1, b = 2], [a = 3, b = 4]}) 中名为 "copied column" 的列。

```
Table.DuplicateColumn(Table.FromRecords({[a = 1, b = 2], [a = 3, b = 4]}), "a", "copied column")
Table.FromRecords({ [
a = 1,
b = 2,
#"copied column" = 1
], [
a = 3,
b = 4,
#"copied column" = 3
]
}, {
"a",
"b",
"copied column"
})
```

● Table.ExpandListColumn

给定 table 后(其中 column 是值列表)，针对每个值将列表拆分为一行。将在每个创建的新行中复制其他列的值。

Table.ExpandListColumn(table as table, column as text) as table

拆分表中的列表列 [Name]。

```
Table.ExpandListColumn(Table.FromRecords({[Name= {"Bob", "Jim", "Paul"}, Discount = .15]}), "Name")
Table.FromRecords({[Name="Bob", Discount=0.15],[Name="Jim", Discount=0.15],[Name="Paul", Discount=0.15]})
```

● Table.ExpandRecordColumn

给定输入 table 中的 column 条记录后，创建一个表，其中针对记录中的每个字段都有一列。也可以指定 newColumnNames ，以确保新表中的各列具有唯一名称。

table : 原始表以及要扩展的记录列。

column : 要扩展的列。

fieldNames : 要扩展到表中各列的字段列表。

newColumnNames：给予新列的列名的列表。新列名不能与新表中的任何列重复。

Table.ExpandRecordColumn(table as table, column as text, fieldNames as list, optional newColumnNames as nullable list) as table

将表 ({[a = [aa = 1, bb = 2, cc = 3], b = 2]}) 中的列 [a] 扩展为 3 列 "aa"、"bb" 和 "cc"。

```
Table.ExpandRecordColumn(Table.FromRecords({[a = [aa = 1, bb = 2, cc = 3], b = 2]}), "a", {"aa", "bb", "cc"})
Table.FromRecords({[aa = 1,
bb = 2,
cc = 3,
b = 2]}, {
"aa",
"bb",
"cc",
"b"
})
```

● Table.ExpandTableColumn

将 table [column] 中的表扩展为多个行和列。 columnNames 用于从内部表中选择要扩展的列。指定 newColumnNames 以避免现有列与新列之间的冲突。

Table.ExpandTableColumn(table as table, column as text, columnNames as list, optional newColumnNames as nullable list) as table

将表 ({[t = {[a=1, b=2, c=3], [a=2,b=4,c=6]}, b = 2]}) 的 [a] 中的表列扩展为 3 列 [t.a]、[t.b] 和 [t.c]。

```
Table.ExpandTableColumn(Table.FromRecords({[t = Table.FromRecords({[a=1, b=2, c=3],[a=2,b=4,c=6]}), b = 2]}), "t", {"a","b","c"}, {"t.a","t.b","t.c"})
Table.FromRecords({[t.a = 1, t.b = 2, t.c = 3, b = 2],
[t.a = 2, t.b = 4, t.c = 6, b = 2]}, {
"t.a",
"t.b",
"t.c",
"b"
})
```

● Table.FillDown

从指定的 table 中返回一个表，其中前一个单元的值传播到指定的 columns 下值为 Null 的单元。

Table.FillDown(table as table, columns as list) as table

从表返回一个表，其中，列 [Place] 中的 null 值使用这些值上方的值填充。

Table.FillDown(Table.FromRecords({[Place=1, Name="Bob"], [Place=null, Name="John"], [Place=2, Name="Brad"], [Place=3, Name="Mark"], [Place=null, Name="Tom"], [Place=null, Name="Adam"]}), {"Place"})
Table.FromRecords({[Place=1,Name="Bob"],[Place=1,Name="John"],[Place=2,Name="Brad"],[Place=3,Name="Mark"],[Place=3,Name="Tom"],[Place=3,Name="Adam"]})

● Table.FillUp

从指定的 table 中返回一个表，其中下一个单元的值传播到指定的 columns 上面值为 Null 的单元。

Table.FillUp(table as table, columns as list) as table

从表返回一个表，其中，列 [Column2] 中的 null 值使用这些值下方的值填充。

Table.FillUp(Table.FromRecords({[Column1 = 1, Column2 = 2], [Column1 = 3, Column2 = null], [Column1 = 5, Column2 = 3]}), {"Column2"})
Table.FromRecords({[Column1=1,Column2=2],[Column1=3,Column2=3],[Column1=5,Column2=3]})

● Table.FilterWithDataTable

Table.FilterWithDataTable

Table.FilterWithDataTable(table as table, dataTableIdentifier as text) as any

● Table.FindText

返回表 table 中包含文本 text 的行。如果找不到文本，则返回空表。

Table.FindText(table as table, text as text) as table

查找表中包含 "Bob" 的行。

Table.FindText(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), "Bob")
Table.FromRecords({[CustomerID=1,Name="Bob",Phone="123-4567"]})

● Table.First

返回 table 的第一行，或如果表为空，则返回可选默认值 default 。

Table.First(table as table, optional default as any) as any

查找表的第一行。

Table.First(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"] , [CustomerID = 3, Name = "Paul", Phone = "543-7890"]}))
[CustomerID = 1, Name = "Bob", Phone = "123-4567"]

查找表 ({}) 的第一行，或如果为空，则返回 [a = 0, b = 0]。

```
Table.First(Table.FromRecords({}), [a = 0, b = 0])
[a = 0, b = 0]
```

● Table.FirstN

返回表 table 的前几行，具体取决于 countOrCondition 的值:

如果 countOrCondition 为数字，则将返回多行(从顶部开始)。

如果 countOrCondition 是条件，将返回满足此条件的行，直到行不满足条件为止。

```
Table.FirstN( table as table, countOrCondition as any) as table
```

查找表的前两行。

```
Table.FirstN(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"] , [CustomerID = 3, Name = "Paul", Phone = "543-7890"]}), 2)
Table.FromRecords({[CustomerID=1,Name="Bob",Phone="123-4567"],[CustomerID=2,Name="Jim",Phone="987-6543"]})
```

查找表中 [a] > 0 的前几行。

```
Table.FirstN(Table.FromRecords({[a = 1, b = 2], [a = 3, b = 4], [a = -5, b = -6]}), each [a] > 0)
Table.FromRecords({ [a = 1, b = 2],
[a = 3, b = 4] })
```

● Table.FirstValue

返回表 table 的第一行的第一列或指定的默认值。

```
Table.FirstValue( table as table, optional default as any) as any
```

● Table.FromColumns

从包含嵌套列表以及列名和值的列表 lists 创建类型为 columns 的表。

如果某些列比其他列具有更多值，则当这些列可为 Null 时，缺少的值将使用默认值 "Null" 填充。

```
Table.FromColumns( lists as list, optional columns as any) as table
```

从列表中的客户名称列表返回一个表。客户列表项中的每个值将成为一个行值，每个列表将成为一个列。

```
Table.FromColumns({ {1, "Bob", "123-4567"} , {2, "Jim", "987-6543"}, {3, "Paul", "543-7890"} })
```

Table.FromRecords({[Column1=1,Column2=2,Column3=3],[Column1="Bob",Column2="Jim",Column3="Paul"],[Column1="123-4567",Column2="987-6543",Column3="543-7890"]})

从给定的列列表和列名列表创建一个表。

Table.FromColumns({ {1, "Bob", "123-4567"} , {2, "Jim", "987-6543"}, {3, "Paul", "543-7890"}}, {"CustomerID", "Name", "Phone"})

Table.FromRecords({[CustomerID=1,Name=2,Phone=3],[CustomerID="Bob",Name="Jim",Phone="Paul"],[CustomerID="123-4567",Name="987-6543",Phone="543-7890"]})

创建一个表，其中每行的列数不同。缺失的行值为 null。

Table.FromColumns({ {1, 2, 3}, {4, 5}, {6, 7, 8, 9} }, {"column1", "column2", "column3"})

Table.FromRecords({[column1=1,column2=4,column3=6],[column1=2,column2=5,column3=7],[column1=3,column2=null,column3=8],[column1=null,column2=null,column3=9]})

● Table.FromList

通过将可选的拆分函数 splitter 应用于列表中的每一项，将列表 list 转换为表。默认情况下，假定列表是用逗号分隔的文本值的列表。可选的 columns 可以是列数、列的列表或 TableType。还可以指定可选的 default 和 extraValues 。

Table.FromList(list as list, optional splitter as nullable function, optional columns as any, optional default as any, optional extraValues as nullable number) as table

使用默认的拆分器从列表以及名为 "Letters" 的列创建表。

```
Table.FromList({"a", "b", "c", "d"}, null, {"Letters"})
Table.FromRecords({ [
Letters = "a"
], [
Letters = "b"
], [
Letters = "c"
], [
Letters = "d"
]
}, {
"Letters"
})
```

使用 Record.FieldValues 拆分器以及所生成的具有 "CustomerID" 和 "Name" 作为列名的表，从列表创建一个表。

```
Table.FromList({[CustomerID=1,Name="Bob"],[CustomerID=2,Name="Jim"]} , Record.FieldValues, {"CustomerID", "Name"})
Table.FromRecords({[CustomerID=1,Name="Bob"],[CustomerID=2,Name="Jim"]})
```

● Table.FromPartitions

返回为组合分区表 partitions 集的结果的表。 partitionColumn 是要添加的列名称。列类型默认为 any ，但可以由 partitionColumnType 指定。

```
Table.FromPartitions( partitionColumn as text, partitions as list, optional partitionColumnType as nullable type) as table
```

从列表 {number} 中找到项类型。

```
Table.FromPartitions(
"Year",
{
{
1994,
Table.FromPartitions(
"Month",
{
{
"Jan",
Table.FromPartitions(
"Day",
{
{
1,
#table({"Foo"},{{"Bar"}})
},
{
2,
#table({"Foo"},{{"Bar"}})
}
}
)
},
{
"Feb",
Table.FromPartitions(
```

...

● Table.FromRecords

将记录列表 records 转换为表。

Table.FromRecords(records as list, optional columns as any) as table

使用记录字段名称作为列名称，通过记录创建表。

Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"]})
Table.FromRecords({ [CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"]})

使用键入的列通过记录创建表并选择数字列。

Table.ColumnsOfType(Table.FromRecords({[CustomerID=1, Name="Bob"]}, type table[CustomerID=Number.Type, Name=Text.Type]), {type number})
{"CustomerID"}

● Table.FromRows

从列表 rows 创建一个表，其中该列表的每个元素都是一个包含用于单一行的列值的内部列表。可以为 columns 提供可选的列名列表、表类型或若干列。

Table.FromRows(rows as list, optional columns as any) as table

使用列 [CustomerID] 以及值 {1, 2}、列 [Name] 以及值 {"Bob", "Jim"}、列 [Phone] 以及值 {"123-4567", "987-6543"} 返回一个表。

Table.FromRows({ {1, "Bob", "123-4567"},{2, "Jim", "987-6543"}},{"CustomerID", "Name", "Phone"})
Table.FromRecords({[CustomerID=1, Name="Bob", Phone="123-4567"],[CustomerID=2, Name="Jim", Phone="987-6543"]})

使用列 [CustomerID] 以及值 {1, 2}、列 [Name] 以及值 {"Bob", "Jim"}、列 [Phone] 以及值 {"123-4567", "987-6543"} 返回一个表，其中 [CustomerID] 是数值类型，[Name] 和 [Phone] 是文本类型。

Table.FromRows({{1, "Bob", "123-4567"}, {2, "Jim", "987-6543"}}, type table [CustomerID = number, Name = text, Phone = text])
Table.FromRecords({[CustomerID=1,Name="Bob",Phone="123-4567"],[CustomerID=2,Name="Jim",Phone="987-6543"]})

● Table.FromValue

使用包含提供的值或值列表 value 的列创建表。可选记录参数 options 可以指定为控制以下选项:

DefaultColumnName ：从列表或标量值构造表时使用的列名。

Table.FromValue(value as any, optional options as nullable record) as table

从值 1 创建一个表。

```
Table.FromValue(1)
Table.FromRecords({ [ Value = 1 ] })
----------
```

从列表创建一个表。

```
Table.FromValue({1, "Bob", "123-4567"})
Table.FromRecords({[ Value = 1 ], [Value = "Bob"], [Value="123-4567"]})
----------
```

使用自定义列名从值 1 创建一个表。

```
Table.FromValue(1, [DefaultColumnName = "MyValue"])
Table.FromRecords({ [ MyValue = 1 ] })
```

● Table.Group

按为每行指定的列 key 中的值对 table 的行进行分组。

对于每个组，将构造一条记录，其中包含键列(及其值)以及由 aggregatedColumns 指定的任何聚合列。

注意，如果多个键与比较器匹配，将返回不同的键。此函数无法保证返回固定的行顺序。

或者，也可以指定 groupKind 和 comparer 。

Table.Group(table as table, key as any, aggregatedColumns as list, optional groupKind as nullable number, optional comparer as nullable function) as table

对表进行分组，同时添加一个聚合列 [total]，其中包含价格总和("each List.Sum([price])")。

```
Table.Group(Table.FromRecords({[CustomerID= 1, price = 20], [CustomerID= 2, price = 10], [CustomerID= 2, price = 20], [CustomerID= 1, price = 10], [CustomerID= 3, price = 20], [CustomerID= 3, price = 5]}), "CustomerID", {"total",each List.Sum([price])})
Table.FromRecords({ [CustomerID= 1, total = 30], [CustomerID= 2, total = 30], [CustomerID= 3, total = 25]}, {"CustomerID", "total"})
```

● Table.HasColumns

指示 table 是否包含指定的列 columns 。如果表包含此列或这些列，则返回 true ；否则返回 false 。

Table.HasColumns(table as table, columns as any) as logical

确定表是否具有列 [Name]。

Table.HasColumns(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}),"Name")

true

查找表是否具有列 [Name] 和 [PhoneNumber]。

Table.HasColumns(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}),{"Name", "PhoneNumber"})

false

● Table.InsertRows

返回一个表，其中行列表 rows 插入到 table 的给定位置 offset 。要插入的行中的每列都必须与表的列类型相符。

Table.InsertRows(table as table, offset as number, rows as list) as table

将行插入表中的位置 1。

Table.InsertRows(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"]}), 1, {[CustomerID = 3, Name = "Paul", Phone = "543-7890"]})

Table.FromRecords({[CustomerID=1,Name="Bob",Phone="123-4567"],[CustomerID=3,Name="Paul",Phone="543-7890"],[CustomerID=2,Name="Jim",Phone="987-6543"]})

将两行插入表中的位置 1。

Table.InsertRows(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}), 1, {[CustomerID = 2, Name = "Jim", Phone = "987-6543"],[CustomerID = 3, Name = "Paul", Phone = "543-7890"] })

Table.FromRecords({[CustomerID=1,Name="Bob",Phone="123-4567"],[CustomerID=2,Name="Jim",Phone="987-6543"], [CustomerID=3,Name="Paul",Phone="543-7890"]})

● Table.IsDistinct

指示 table 是否仅包含非重复行(没有重复项)。如果行为非重复行，则返回 true ；否则返回 false 。

可选参数 comparisonCriteria 指定对表中的哪些列进行测试以确定是否具有重复项。如果未指定 comparisonCriteria ，则测试所有列。

Table.IsDistinct(table as table, optional comparisonCriteria as any) as logical

确定表是否为非重复表。

```
Table.IsDistinct(Table.FromRecords({ [CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"] , [CustomerID = 3, Name = "Paul", Phone = "543-7890"] , [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}))
```

true

确定表的列中是否非重复。

```
Table.IsDistinct(Table.FromRecords({ [CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"] , [CustomerID = 3, Name = "Paul", Phone = "543-7890"] , [CustomerID = 5, Name = "Bob", Phone = "232-1550"]}), "Name")
```

false

● Table.IsEmpty

指示 table 是否包含任何行。如果没有行(也即表为空)，则返回 true ； 否则，返回 false 。

Table.IsEmpty(table as table) as logical

确定表是否为空。

```
Table.IsEmpty(Table.FromRecords({[CustomerID =1, Name ="Bob", Phone = "123-4567"],[CustomerID =2, Name ="Jim", Phone = "987-6543"],[CustomerID =3, Name ="Paul", Phone = "543-7890"]}))
```

false

确定表 ({}) 是否为空。

```
Table.IsEmpty(Table.FromRecords({}))
```

true

● Table.Join

根据由 key1 (针对 table1)和 key2 (针对 table2)选择的键列值的等同性联接 table1 的行与 table2 的行。

默认情况下，将执行内联，但可以包含可选的 joinKind 来指定联接类型。选项包括:

JoinKind.Inner
JoinKind.LeftOuter
JoinKind.RightOuter
JoinKind.FullOuter
JoinKind.LeftAnti
JoinKind.RightAnti

可能加入 keyEqualityComparers 的可选集以指定如何比较键列。

Table.Join(table1 as table, key1 as any, table2 as table, key2 as any, optional joinKind as nullable number, optional joinAlgorithm as nullable number, optional keyEqualityComparers as nullable list) as table

根据 [CustomerID] 对两个表执行内部联接。

Table.Join(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), "CustomerID", Table.FromRecords({ [OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0], [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0], [OrderID = 3, CustomerID = 2, Item = "Fishing net", Price = 25.0], [OrderID = 4, CustomerID = 3, Item = "Fish tazer", Price = 200.0], [OrderID = 5, CustomerID = 3, Item = "Bandaids", Price = 2.0], [OrderID = 6, CustomerID = 1, Item = "Tackle box", Price = 20.0], [OrderID = 7, CustomerID = 5, Item = "Bait", Price = 3.25], [OrderID = 8, CustomerID = 5, Item = "Fishing Rod", Price = 100.0], [OrderID = 9, CustomerID = 6, Item = "Bait", Price = 3.25]}), "CustomerID")
Table.FromRecords({[CustomerID=1,Name="Bob",Phone="123-4567",OrderID=1,Item="Fishing rod",Price=100],[CustomerID...

● Table.Keys

Table.Keys

Table.Keys(table as table) as list

● Table.Last

返回 table 的最后一行，或如果表为空，则返回可选默认值 default 。

Table.Last(table as table, optional default as any) as any

查找表的最后一行。

Table.Last(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"]}))

[CustomerID = 3, Name = "Paul", Phone = "543-7890"]

查找表 ({}) 的最后一行，或如果为空，则返回 [a = 0, b = 0]。

```
Table.Last(Table.FromRecords({}), [a = 0, b = 0])
[a = 0, b = 0]
```

● Table.LastN

返回表 table 中的最后几行，具体取决于 countOrCondition 的值:

如果 countOrCondition 为数字，则将返回从位置(结尾 - countOrCondition)开始的多行。

如果 countOrCondition 是条件，将以升序位置返回满足此条件的行，直到行不满足条件为止。

```
Table.LastN( table as table, countOrCondition as any) as table
```

查找表的最后两行。

```
Table.LastN(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"] , [CustomerID = 3, Name = "Paul", Phone = "543-7890"]}), 2)
Table.FromRecords({[CustomerID=2,Name="Jim",Phone="987-6543"],[CustomerID=3,Name="Paul",Phone="543-7890"]})
```

查找表中 [a] > 0 的后几行。

```
Table.LastN(Table.FromRecords({[a = -1, b = -2], [a = 3, b = 4], [a = 5, b = 6]}), each _ [a] > 0)
Table.FromRecords({[a = 3, b = 4],
[a = 5, b = 6]})
```

● Table.MatchesAllRows

指示是否 table 中的所有行都满足给定的 condition 。如果所有行都匹配，则返回 true ；否则返回 false 。

```
Table.MatchesAllRows( table as table, condition as function) as logical
```

确定列 [a] 中的所有行值是否在表中。

```
Table.MatchesAllRows(Table.FromRecords({[a = 2, b = 4], [a = 6, b = 8]}), each Number.Mod([a], 2) = 0 )
true
```

查找在表 ({[a = 1, b = 2], [a = 3, b = 4]}) 中，是否所有行值均为 [a = 1, b = 2]。

```
Table.MatchesAllRows(Table.FromRecords({[a = 1, b = 2], [a = -3, b = 4]}), each _ = [a = 1, b = 2])
false
```

● Table.MatchesAnyRows

指示是否 table 中的任何行都满足给定的 condition 。如果任何行都匹配，则返回 true ；否则返回 false 。

```
Table.MatchesAnyRows( table as table, condition as function) as logical
```

确定列 [a] 中的任何行值是否在表 ({[a = 2, b = 4], [a = 6, b = 8]}) 中。

```
Table.MatchesAnyRows(Table.FromRecords({[a = 1, b = 4], [a = 3, b = 8]}), each Number.Mod([a], 2) = 0 )
false
----------
```

确定表 ({[a = 1, b = 2], [a = 3, b = 4]}) 中任何行值是否均为 [a = 1, b = 2]。

```
Table.MatchesAnyRows(Table.FromRecords({[a = 1, b = 2], [a = -3, b = 4]}), each _ = [a = 1, b = 2])
true
```

● Table.Max

在给定 comparisonCriteria 的情况下，返回 table 中的最大值行。如果表为空，则返回可选的 default 值。

```
Table.Max( table as table, comparisonCriteria as any, optional default as any) as any
```

查找表 ({[a = 2, b = 4], [a = 6, b = 8]}) 的列 [a] 中具有最大值的行。

```
Table.Max(Table.FromRecords({[a = 2, b = 4], [a = 6, b = 8]}), "a")
[a = 6, b = 8]
----------
```

查找表 ({}) 的列 [a] 中具有最大值的行。如果为空，则返回 -1。

```
Table.Max(#table({"a"},{}), "a", -1)
-1
```

● Table.MaxN

在给定 comparisonCriteria 的情况下，返回 table 中的最大值行。

在对行排序后，必须指定 countOrCondition 参数以进一步筛选结果。注意，排序算法无法保证固定的排序结果。 countOrCondition 参数可以采用多种格式:

如果指定一个数，则返回以升序排序的、最多包含 countOrCondition 项的列表。

如果指定条件，则返回最初满足该条件的项列表。如果某个项不满足该条件，则不再考虑其他项。

Table.MaxN(table as table, comparisonCriteria as any, countOrCondition as any) as table

查找表中在列 [a] 中具有最大值且条件 [a] > 0 的行。在应用筛选器之前，先对行排序。

```
Table.MaxN(Table.FromRecords({[a = 2, b = 4], [a = 0, b = 0], [a = 6, b = 2]}), "a", each [a] > 0)
Table.FromRecords({[a = 6, b = 2],
[a = 2, b = 4]})
----------
```

查找表中在列 [a] 中具有最大值且条件 [b] > 0 的行。在应用筛选器之前，先对行排序。

```
Table.MaxN(Table.FromRecords({[a = 2, b = 4], [a = 8, b = 0], [a = 6, b = 2]}), "a", each [b] > 0)
Table.FromRecords({})
```

● Table.Min

在给定 comparisonCriteria 的情况下，返回 table 中的最小值行。如果表为空，则返回可选的 default 值。

Table.Min(table as table, comparisonCriteria as any, optional default as any) as any

查找表的列 [a] 中具有最小值的行。

```
Table.Min(Table.FromRecords({[a = 2, b = 4], [a = 6, b = 8]}), "a")
[a = 2, b = 4]
----------
```

查找表的列 [a] 中具有最小值的行。如果为空，则返回 -1。

```
Table.Min(#table({"a"},{}), "a", -1)
-1
```

● Table.MinN

在给定 comparisonCriteria 的情况下，返回 table 中的最小值行。在对行排序后，必须指定 countOrCondition 参数以进一步筛选结果。注意，排序算法无法保证固定的排序结果。

countOrCondition 参数可以采用多种格式:

如果指定一个数，则返回以升序排序的、最多包含 countOrCondition 项的列表。

如果指定条件，则返回最初满足该条件的项列表。如果某个项不满足该条件，则不再考虑其他项。

Table.MinN(table as table, comparisonCriteria as any, countOrCondition as any) as table

查找表中在列 [a] 中具有最小值且条件 [a] < 3 的行。在应用筛选器之前，先对行排序。

```
Table.MinN(Table.FromRecords({[a = 2, b = 4], [a = 0, b = 0], [a = 6, b = 4]}), "a", each [a] < 3)
Table.FromRecords({[a = 0, b = 0],
[a = 2, b = 4]})
----------
```

查找表中在列 [a] 中具有最小值且条件 [b] < 0 的行。在应用筛选器之前，先对行排序。

```
Table.MinN(Table.FromRecords({[a = 2, b = 4], [a = 8, b = 0], [a = 6, b = 2]}), "a", each [b] < 0)
Table.FromRecords({})
```

● Table.NestedJoin

根据由 key1 (针对 table1)和 key2 (针对 table2)选择的键列值的等同性联接 table1 的行与 table2 的行。将结果输入名为 newColumnName 的列。

可选的 joinKind 指定要执行的联接类型。默认情况下，如果未指定 joinKind ，则执行左外部联接。

可能加入 keyEqualityComparers 的可选集以指定如何比较键列。

Table.NestedJoin(table1 as table, key1 as any, table2 as any, key2 as any, newColumnName as text, optional joinKind as nullable number, optional keyEqualityComparers as nullable list) as table

● Table.Partition

根据 column 和 hash 函数的值，将 table 分区为一组 groups 个表。

hash 函数应用于 column 行的值，以获取该行的哈希值。哈希值模数 groups 确定要将该行放入所返回的哪个表中。

table : 要分区的表。

column： 要执行哈希运算以确定该行位于返回的哪个表中的列。

groups： 要对输入表进行分区的表数。

hash： 应用以获取哈希值的函数。

Table.Partition(table as table, column as text, groups as number, hash as function) as list

使用列的值作为哈希函数，基于列 [a] 将表 ({[a = 2, b = 4], [a = 6, b = 8], [a = 2, b = 4], [a = 1, b = 4]}) 分区为 2 个表。

```
Table.Partition(Table.FromRecords({[a = 2, b = 4], [a = 1, b = 4], [a = 2, b = 4], [a = 1, b = 4]}), "a", 2, each _)
{ Table.FromRecords({[a = 2, b = 4], [a = 2, b = 4]}, {
"a",
"b"
}),
Table.FromRecords({[a = 1, b = 4], [a = 1, b = 4]}, {
"a",
"b"
}) }
```

● Table.PartitionValues

返回有关如何对表进行分区的信息。 此时将返回一个表，其中的每列为原始表中的一个分区列，每行对应于原始表中的一个分区。

Table.PartitionValues(table as table) as table

● Table.Pivot

给定一对表示属性-值对的列，将属性列中的数据旋转为列标题。

Table.Pivot(table as table, pivotValues as list, attributeColumn as text, valueColumn as text, optional aggregationFunction as nullable function) as table

选取表 ({ [key = "x", attribute = "a", value = 1], [key = "x", attribute = "c", value = 3], [key = "y", attribute = "a", value = 2], [key = "y", attribute = "b", value = 4] }) 的属性列中的值 "a"、"b" 和 "c"，并将它们透视为其自己的列。

```
Table.Pivot(Table.FromRecords({ [ key = "x", attribute = "a", value = 1 ], [ key = "x", attribute = "c", value = 3 ], [ key = "y", attribute = "a", value = 2 ], [ key = "y", attribute = "b", value = 4 ] }), { "a", "b", "c" }, "attribute", "value")
Table.FromRecords({[ key = "x", a = 1, b = null, c = 3 ], [ key = "y", a = 2, b = 4, c = null ]})
```

选取表 ({ [key = "x", attribute = "a", value = 1], [key = "x", attribute = "c", value = 3], [key = "x", attribute = "c", value = 5], [key = "y", attribute = "a", value = 2], [key = "y", attribute = "b", value = 4] }) 的属性列中的值 "a"、"b" 和 "c"，并将它们透视为其自己的列。 键 "x" 的属性 "c" 具有多个与其关联的值，因此使用函数 List.Max 来解决冲突。

Table.Pivot(Table.FromRecords({ [key = "x", attribute = "a", value = 1], [key = "x", attribute = "...

● Table.PositionOf

返回 row 在指定的 table 中第一次出现的行位置。如果找不到该值，则返回 -1。

table : 输入表。

row : 表中要查找其位置的行。

occurrence : [可选] 指定要返回的行的出现次数。

equationCriteria : [可选] 控制表行之间的比较。

Table.PositionOf(table as table, row as record, optional occurrence as any, optional equationCriteria as any) as any

查找 [a = 2, b = 4] 在表 ({[a = 2, b = 4], [a = 6, b = 8], [a = 2, b = 4], [a = 1, b = 4]}) 中第一次出现的位置。

Table.PositionOf(Table.FromRecords({[a = 2, b = 4], [a = 1, b = 4], [a = 2, b = 4], [a = 1, b = 4]}), [a = 2, b = 4])

0

查找 [a = 2, b = 4] 在表 ({[a = 2, b = 4], [a = 6, b = 8], [a = 2, b = 4], [a = 1, b = 4]}) 中第二次出现的位置。

Table.PositionOf(Table.FromRecords({[a = 2, b = 4], [a = 1, b = 4], [a = 2, b = 4], [a = 1, b = 4]}), [a = 2, b = 4], 1)

2

查找 [a = 2, b = 4] 在表 ({[a = 2, b = 4], [a = 6, b = 8], [a = 2, b = 4], [a = 1, b = 4]}) 中所有出现的位置。

Table.PositionOf(Table.FromRecords({[a = 2, b = 4], [a = 1, b = 4], [a = 2, b = 4], [a = 1, b = 4]}), [a = 2, b = 4], Occurrence.All)

{0, 2}

● Table.PositionOfAny

返回 rows 的列表在 table 中第一次出现的行位置。如果找不到该值，则返回 -1。

table : 输入表。

rows : 表中要查找其位置的行列表。

occurrence : [可选] 指定要返回的行的出现次数。

equationCriteria : [可选] 控制表行之间的比较。

Table.PositionOfAny(table as table, rows as list, optional occurrence as nullable number, optional equationCriteria as any) as any

查找 [a = 2, b = 4] 或 [a = 6, b = 8] 在表 ({[a = 2, b = 4], [a = 6, b = 8], [a = 2, b = 4], [a = 1, b = 4]}) 中第一次出现的位置。

```
Table.PositionOfAny(Table.FromRecords({[a = 2, b = 4], [a = 1, b = 4], [a = 2, b = 4], [a = 1, b = 4]}), {[a = 2, b = 4], [a = 6, b = 8]})
0
----------
```

查找 [a = 2, b = 4] 或 [a = 6, b = 8] 在表 ({[a = 2, b = 4], [a = 6, b = 8], [a = 2, b = 4], [a = 1, b = 4]} 中所有出现的位置。

```
Table.PositionOfAny(Table.FromRecords({[a = 2, b = 4], [a = 6, b = 8], [a = 2, b = 4], [a = 1, b = 4]}), {[a = 2, b = 4], [a = 6, b = 8]}, Occurrence.All)
{0, 1, 2}
```

● Table.PrefixColumns

返回一个表，其中来自所提供的 table 中的所有列名均以给定的文本 prefix 为前缀，另加一个采用格式 prefix .ColumnName 的句点。

Table.PrefixColumns(table as table, prefix as text) as table

为表中的列加前缀 "MyTable"。

```
Table.PrefixColumns(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}), "MyTable")
Table.FromRecords({[MyTable.CustomerID=1,MyTable.Name="Bob", MyTable.Phone="123-4567"]})
```

● Table.Profile

返回 table 中列的配置文件。

为每一列返回以下信息(如果适用):

最小值

最大值

平均值

标准偏差

计数

Null 计数

非重复计数

Table.Profile(table as table) as table

● Table.PromoteHeaders

将第一行值升级为新的列标题(即列名)。默认情况下，仅将文本或数值升级为标题。有效选项:

PromoteAllScalars ：如果设置为 true ，则使用 Culture (如果已指定，或当前文档区域设置)将第一行中的所有标量值升级为标题。

对于无法转换为文本的值，将使用默认列名。

Culture ：区域性名称，指定数据的区域性。

Table.PromoteHeaders(table as table, optional options as nullable record) as table

升级表中的第一行值。

```
Table.PromoteHeaders(Table.FromRecords({[Column1 = "CustomerID", Column2 = "Name", Column3 = #date(1980,1,1)], [Column1 = 1, Column2 = "Bob", Column3 = #date(1980,1,1)]}))
Table.FromRecords({[CustomerID = 1,Name = "Bob", Column3 = #date(1980,1,1)]})
----------
```

将表的第一行中的所有标量升级为标题。

Table.PromoteHeaders(Table.FromRecords({[Rank = 1, Name = "Name", Date = #date(1980,1,1)],[Rank = 1, Name = "Bob", Date = #date(1980,1,1)]}), [PromoteAllScalars = true, Culture = "en-US"])

Table.FromRecords({[1 = 1, Name = "Bob", #"1/1/1980" = #date(1980, 1, 1)]})

● Table.Range

以指定的 offset 开始返回 table 中的行。可选参数 count 指定要返回的行数。默认情况下，将返回偏移量之后的所有行。

Table.Range(table as table, offset as number, optional count as nullable number) as table

返回表中以偏移量 1 开始的所有行。

Table.Range(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), 1)

Table.FromRecords({[CustomerID=2,Name="Jim",Phone="987-6543"],[CustomerID=3,Name="Paul",Phone="543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]})

返回表中以偏移量 1 开始的一行。

Table.Range(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), 1, 1)

Table.FromRecords({[CustomerID=2,Name="Jim",Phone="987-6543"]})

● Table.RemoveColumns

从提供的 columns 删除指定的 table 。

如果此列不存在，将引发异常，除非可选参数 missingField 指定备用值(例如，MissingField.UseNull 或 MissingField.Ignore)。

Table.RemoveColumns(table as table, columns as any, optional missingField as nullable number) as table

从表中删除列 [Phone]。

Table.RemoveColumns(Table.FromRecords({[CustomerID=1, Name="Bob", Phone = "123-4567"]}), "Phone")

Table.FromRecords({[CustomerID=1,Name="Bob"]})

从表中删除列 [Address]。如果该列不存在，则引发一个错误。

Table.RemoveColumns(Table.FromRecords({[CustomerID=1, Name="Bob", Phone = "123-4567"]}), "Address")

[Expression.Error] The field 'Address' of the record was not found.

● Table.RemoveFirstN

返回一个表，该表不包含表 table 的指定数量的前几行 countOrCondition 。

删除的行数依赖于可选参数 countOrCondition 。

如果忽略 countOrCondition ，则只删除第一行。

如果 countOrCondition 为数字，则将删除该数字那么多的行(从顶部开始)。

如果 countOrCondition 是条件，将删除满足此条件的行，直到行不满足条件为止。

Table.RemoveFirstN(table as table, countOrCondition as any) as table

删除表的第一行。

Table.RemoveFirstN(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), 1)

Table.FromRecords({[CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]})

删除表的前两行。

Table.RemoveFirstN(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), 2)

Table.FromRecords({[CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]})

删除表中 [CustomerID] <=2 的前几行。

Table.RemoveFirstN(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = ...

● Table.RemoveLastN

返回一个表，该表不包含表 table 的最后 countOrCondition 行。

删除的行数依赖于可选参数 countOrCondition 。

如果忽略 countOrCondition ，则只删除最后一行。

如果 countOrCondition 为数字，则将删除该数字那么多的行(从底部开始)。

如果 countOrCondition 是条件，将删除满足此条件的行，直到行不满足条件为止。

Table.RemoveLastN(table as table, optional countOrCondition as any) as table

删除表的最后一行。

```
Table.RemoveLastN(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"],[CustomerID = 2, Name = "Jim", Phone = "987-6543"],[CustomerID = 3, Name = "Paul", Phone = "543-7890"],[CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), 1)
Table.FromRecords({[CustomerID=1,Name="Bob",Phone="123-4567"],[CustomerID=2,Name="Jim",Phone="987-6543"],[CustomerID=3,Name="Paul",Phone="543-7890"]})
----------
```

删除表中 [CustomerID] > 2 的最后几行。

```
Table.RemoveLastN(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"],[CustomerID = 2, Name = "Jim", Phone = "987-6543"],[CustomerID = 3, Name = "Paul", Phone = "543-7890"],[CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), each [CustomerID] >= 2)
Table.FromRecords({[CustomerID=1,Name="Bob",Phone="123-4567"]})
```

● Table.RemoveMatchingRows

从 table 中删除出现的所有指定 rows 。

可以指定一个可选参数 equationCriteria ，以控制表各行之间的比较。

Table.RemoveMatchingRows(table as table, rows as list, optional equationCriteria as any) as table

从表 ({[a = 1, b = 2], [a = 3, b = 4], [a = 1, b = 6]}) 中删除其中具有 [a = 1] 的任何行。

```
Table.RemoveMatchingRows(Table.FromRecords({[a = 1, b = 2], [a = 3, b = 4], [a = 1, b = 6]}), {[a = 1]}, "a")
Table.FromRecords({[a = 3, b = 4]}, {
"a",
"b"
})
```

● Table.RemoveRows

从 table 的开头，以指定的 offset 开始删除 count 行。如果没有提供 count 参数，则使用默认计数 1。

Table.RemoveRows(table as table, offset as number, optional count as nullable number) as table

从表中删除第一行。

Table.RemoveRows(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), 0)
Table.FromRecords ({[CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]})

删除位于表的位置 1 的行。

Table.RemoveRows(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), 1)
Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]})

删除表中从位置 1 开始的两行。

Table.RemoveRows(Table.FromRecords({[CustomerID = 1, Name = "Bob", Pho...

● Table.RemoveRowsWithErrors

返回一个表，其中已从输入表中删除了在至少一个单元中包含错误的行。如果指定了列列表，则只检查指定列中的单元内是否有错误。

Table.RemoveRowsWithErrors(table as table, optional columns as nullable list) as table

从第一行中删除错误值。

Table.RemoveRowsWithErrors(Table.FromRecords({[Column1=...],[Column1=2], [Column1=3]}))
Table.FromRecords({[Column1=2], [Column1=3]})

● Table.RenameColumns

对表 table 中的列执行给定的重命名。一个替换操作 renames 由两个值的列表以及某个列表中提供的旧列名和新列名组成。

如果此列不存在，将引发异常，除非可选参数 missingField 指定备用值(例如，MissingField.UseNull 或 MissingField.Ignore)。

Table.RenameColumns(table as table, renames as list, optional missingField as nullable number) as table

在表中将列名 "CustomerNum" 替换为 "CustomerID"。

Table.RenameColumns(Table.FromRecords({[CustomerNum=1, Name="Bob", Phone = "123-4567"]}), {"CustomerNum", "CustomerID"})
Table.FromRecords({[CustomerID=1,Name="Bob", Phone="123-4567"]})

在表中将列名 "CustomerNum" 替换为 "CustomerID"，并将 "PhoneNum" 替换为 "Phone"。

Table.RenameColumns(Table.FromRecords({[CustomerNum=1, Name="Bob", PhoneNum = "123-4567"]}), {{"CustomerNum", "CustomerID"}, {"PhoneNum", "Phone"}})
Table.FromRecords({[CustomerID=1,Name="Bob", Phone="123-4567"]})

在表中将列名 "NewCol" 替换为 "NewColumn"，如果列不存在，则忽略。

Table.RenameColumns(Table.FromRecords({[CustomerID=1, Name="Bob", Phone = "123-4567"]}), {"NewCol", "NewColumn"}, MissingField.Ignore)
Table.FromRecords({[CustomerID=1,Name="Bob", Phone="123-4567"]})

● Table.ReorderColumns

从输入 table 返回一个表，其中的列遵循由 columnOrder 指定的顺序。列表中未指定的列将不重新排序。

如果此列不存在，将引发异常，除非可选参数 missingField 指定备用值(例如，MissingField.UseNull 或 MissingField.Ignore)。

Table.ReorderColumns(table as table, columnOrder as list, optional missingField as nullable number) as table

切换表中列 [Phone] 和 [Name] 的顺序。

Table.ReorderColumns(Table.FromRecords({[CustomerID=1, Phone = "123-4567", Name ="Bob"]}), {"Name","Phone"})
Table.FromRecords({[CustomerID=1,Name="Bob", Phone="123-4567"]})

切换表中列 [Phone] 和 [Address] 的顺序或使用 "MissingField.Ignore"。这不会更改表，因为列 [Address] 不存在。

Table.ReorderColumns(Table.FromRecords({[CustomerID=1, Name = "Bob", Phone = "123-4567"]}), {"Phone", "Address"}, MissingField.Ignore)
Table.FromRecords({[CustomerID=1,Name="Bob", Phone="123-4567"]})

● Table.Repeat

从输入 table 返回一个表，其中的列重复了指定的 count 次。

Table.Repeat(table as table, count as number) as table

对表中的行重复两次。

```
Table.Repeat(Table.FromRecords({[a = 1, b = "hello"], [a = 3, b = "world"]}), 2)
Table.FromRecords({ [
a = 1,
b = "hello"
], [
a = 3,
b = "world"
], [
a = 1,
b = "hello"
], [
a = 3,
b = "world"
]
}, {
"a",
"b"
})
```

● Table.ReplaceErrorValues

使用 errorReplacement 列表中的新值替换 table 指定列中的错误值。列表的格式为 {{column1, value1}, …}。每列可能只有一个替换值，为列进行多于一次的指定将导致错误。

Table.ReplaceErrorValues(table as table, errorReplacement as list) as table

在表中将错误值替换为文本 "world"。

```
Table.ReplaceErrorValues(Table.FromRows({{1,"hello"},{3,...}}, {"A","B"}), {"B", "world"})
Table.FromRecords({[A = 1, B = "hello"],
[A = 3, B = "world"]}, {
"A",
"B"
})
```

在表中将 A 列中的错误值替换为文本 "hello"，将 B 列中的错误值替换为文本 "world"。

```
Table.ReplaceErrorValues(Table.FromRows({{..., ...},{1,2}}, {"A","B"}), {{"A", "hello"}, {"B", "world"}})
Table.FromRecords({[A = "hello", B = "world"],
[A = 1, B = 2]}, {
"A",
"B"
```

```
})
```

● Table.ReplaceKeys

Table.ReplaceKeys

```
Table.ReplaceKeys( table as table, keys as list) as table
```

● Table.ReplaceMatchingRows

使用提供的行替换 table 中所有指定的行。要替换的行以及替换项在 replacements 中使用 {old, new} 格式指定。

可以指定一个可选 equationCriteria 参数，以控制表各行之间的比较。

```
Table.ReplaceMatchingRows( table as table, replacements as list, optional equationCriteria as any) as table
```

```
在表中将行 [a = 1, b = 2] 和 [a = 2, b = 3] 替换为 [a = -1, b = -2] 和 [a = -2, b = -3]。

Table.ReplaceMatchingRows(Table.FromRecords({[a = 1, b =2], [a = 2, b = 3], [a = 3, b = 4], [a = 1, b = 2]}),{ {[a = 1, b = 2], [a = -1, b = -2]}, {[a = 2, b = 3], [a = -2, b = -3]} })
Table.FromRecords({ [
a = -1,
b = -2
], [
a = -2,
b = -3
], [
a = 3,
b = 4
], [
a = -1,
b = -2
] }, {
"a",
"b"
})
```

● Table.ReplaceRelationshipIdentity

Table.ReplaceRelationshipIdentity

```
Table.ReplaceRelationshipIdentity( value as any, identity as text) as any
```

● Table.ReplaceRows

在输入 table 中使用指定的 rows ，从 offset 之后开始替换指定的行数 count 。 rows 参数是记录列表。

table : 要在其中执行替换的表。

offset : 在进行替换之前要跳过的行数。

count : 要替换的行数。

rows : 要在由 offset 指定的位置插入到 table 中的行记录的列表。

Table.ReplaceRows(table as table, offset as number, count as number, rows as list) as table

从位置 1 开始，替换 3 行。

```
Table.ReplaceRows(Table.FromRecords({[Column1=1], [Column1=2], [Column1=3], [Column1=4], [Column1=5]}), 1, 3, {[Column1=6], [Column1=7]})
Table.FromRecords({[Column1=1],[Column1=6],[Column1=7],[Column1=5]})
```

● Table.ReplaceValue

在 table 的指定列中将 oldValue 替换为 newValue 。

Table.ReplaceValue(table as table, oldValue as any, newValue as any, replacer as function, columnsToSearch as list) as table

在表中将文本 "goodbye" 替换为文本 "world"。

```
Table.ReplaceValue(Table.FromRecords({[a = 1, b = "hello"], [a = 3, b = "goodbye"]}), "goodbye", "world", Replacer.ReplaceText, {"b"})
Table.FromRecords({[a = 1, b = "hello"],
[a = 3, b = "world"]}, {
"a",
"b"
})
----------
```

在表中将文本 "ur" 替换为文本 "or"。

```
Table.ReplaceValue(Table.FromRecords({[a = 1, b = "hello"], [a = 3, b = "wurld"]}), "ur", "or", Replacer.ReplaceText, {"b"})
Table.FromRecords({ [a = 1, b = "hello"],
[a = 3, b = "world"]}, {
"a",
"b"
})
```

● Table.ReverseRows

从输入 table 返回一个表，其中的行遵循相反顺序。

Table.ReverseRows(table as table) as table

使表中的行按相反顺序排列。

Table.ReverseRows(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}))
Table.FromRecords({[CustomerID=4,Name="Ringo",Phone="232-1550"],[CustomerID=3,Name="Paul",Phone="543-7890"],[CustomerID=2,Name="Jim",Phone="987-6543"],[CustomerID=1,Name="Bob",Phone="123-4567"]})

● Table.RowCount

返回表 table 中的行数。

Table.RowCount(table as table) as number

查找表中的行数。

Table.RowCount(Table.FromRecords({[CustomerID =1, Name ="Bob", Phone = "123-4567"],[CustomerID =2, Name ="Jim", Phone = "987-6543"],[CustomerID =3, Name ="Paul", Phone = "543-7890"]}))
3

● Table.Schema

返回一个表，该表描述 table 的列。

该表中的各行分别描述 table 的某一列的属性:

<table>

列名	描述
Name	该列的名称。
Position	

该列在 table 中的位置(从 0 开始)。

TypeName

该列的类型名称。

Kind

该列的类型种类。

IsNullable

该列是否可以包含 null 值。

NumericPrecisionBase

NumericPrecision 和 NumericScale 字段的数值基数(例如，base-2、base-10)。

NumericPrecision

NumericPrecisionBase 指定的基数中数值列的精度。这是此类型的值可以表示的最大位数(包括小数位)。

NumericScale

NumericPrecisionBase 指定的基数中数值列的位数。这是此类型的值的小数部分的位数。值<...

Table.Schema(table as table) as table

● Table.SelectColumns

返回只含有指定 columns 的 table 。

table : 提供的表。

columns：要从表 table 中返回的列的列表。返回的表中的列以 columns 中列出的顺序排列。

missingField：(可选) 如果此列不存在，要执行什么操作。 示例: MissingField.UseNull 或 MissingField.Ignore 。

Table.SelectColumns(table as table, columns as any, optional missingField as nullable number) as table

只包含列 [Name]。

```
Table.SelectColumns(Table.FromRecords({
[CustomerID = 1, Name = "Bob", Phone = "123-4567"],
[CustomerID = 2, Name = "Jim", Phone = "987-6543"] ,
[CustomerID = 3, Name = "Paul", Phone = "543-7890"] ,
[CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
}), "Name")
Table.FromRecords({
[Name = "Bob"],
[Name = "Jim"] ,
[Name = "Paul"] ,
[Name = "Ringo"]
})
```

只包含列 [CustomerID] 和列 [Name]。

```
Table.SelectColumns(Table.FromRecords({[CustomerID=1, Name="Bob", Phone = "123-4567"]}), {"CustomerID", "Name"})
Table.FromRecords({[CustomerID=1, Name="Bob"]})
```

如果包含的列没有退出，则默认结果为错误。

```
Table.SelectColumns(Table.FromRecords({[CustomerID=1, Name="Bob", Phone = "123-4567"]}), "NewColumn")
[Expression.Error] The field 'NewColumn' of the record wasn't found.
```

如果包含的列没有退出，选项 MissingField.UseNull 将创建包含 null 值的列。

```
Table.SelectColumns(Table.FromRecords({[CustomerID=1, Name = "Bob", Phone = "123-4567" ]}), {"Cust...
```

- **Table.SelectRows**

从 table 返回与选择 condition 匹配的行的表。

Table.SelectRows(table as table, condition as function) as table

选择表中的行，其中 [CustomerID] 列中的值大于 2。

```
Table.SelectRows(Table.FromRecords({
[CustomerID = 1, Name = "Bob", Phone = "123-4567"],
[CustomerID = 2, Name = "Jim", Phone = "987-6543"] ,
[CustomerID = 3, Name = "Paul", Phone = "543-7890"] ,
[CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
}), each [CustomerID] > 2)
Table.FromRecords({
[CustomerID = 3, Name = "Paul", Phone = "543-7890"] ,
[CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
})
```

选择表中的行，其中名称不包含 "B"。

```
Table.SelectRows(Table.FromRecords({
[CustomerID = 1, Name = "Bob", Phone = "123-4567"],
[CustomerID = 2, Name = "Jim", Phone = "987-6543"] ,
[CustomerID = 3, Name = "Paul", Phone = "543-7890"] ,
[CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
}), each not Text.Contains([Name], "B"))
Table.FromRecords({
[CustomerID = 2, Name = "Jim", Phone = "987-6543"] ,
[CustomerID = 3, Name = "Paul", Phone = "543-7890"] ,
[CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
})
```

● Table.SelectRowsWithErrors

返回一个表，其中只具有输入表中的在至少一个单元中包含错误的那些行。如果指定了列列表，则只检查指定列中的单元内是否有错误。

Table.SelectRowsWithErrors(table as table, optional columns as nullable list) as table

选择其行中包含错误的客户的名称。

```
Table.SelectRowsWithErrors(Table.FromRecords({
[CustomerID =..., Name = "Bob", Phone = "123-4567"],
[CustomerID = 2, Name = "Jim", Phone = "987-6543"] ,
[CustomerID = 3, Name = "Paul", Phone = "543-7890"] ,
[CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
}))[Name]
```

{"Bob"}

● Table.SingleRow

返回一行 table 中的单一行。如果 table 具有多行，则引发异常。

Table.SingleRow(table as table) as record

返回表中的单行。

Table.SingleRow(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"]}))
[CustomerID = 1, Name = "Bob", Phone = "123-4567"]

● Table.Skip

返回一个表，该表不包含表 table 的指定数量的前几行 countOrCondition 。

跳过的行数依赖于可选参数 countOrCondition 。

如果忽略 countOrCondition ，则只跳过第一行。

如果 countOrCondition 为数字，则将跳过多行(从顶部开始)。

如果 countOrCondition 是条件，将跳过满足此条件的行，直到行不满足条件为止。

Table.Skip(table as table, countOrCondition as any) as table

跳过表的第一行。

Table.Skip(Table.FromRecords({ [CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"] , [CustomerID = 3, Name = "Paul", Phone = "543-7890"] , [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), 1)
Table.FromRecords({[CustomerID=2,Name="Jim",Phone="987-6543"],[CustomerID=3,Name="Paul",Phone="543-7890"],[CustomerID=4,Name="Ringo",Phone="232-1550"]})

跳过表的前两行。

Table.Skip(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"],[CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"], [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]}), 2)
Table.FromRecords({[CustomerID=3,Name="Paul",Phone="543-7890"],[CustomerID=4,Name="Ringo",Phone="232-1550"]})

跳过表中 [Price] > 25 的前几行。

Table.Skip(Table.FromRecords({[OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0], [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0], [OrderID ...

● Table.Sort

使用一个或多个列名的列表和可选的 comparisonCriteria (格式为 { { col1, comparisonCriteria }, {col2} })对 table 排序。

```
Table.Sort( table as table, comparisonCriteria as any) as table
```

```
在列 "OrderID" 上对表排序。
Table.Sort(Table.FromRecords({[OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0], [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0], [OrderID = 3, CustomerID = 2, Item = "Fishing net", Price = 25.0], [OrderID = 4, CustomerID = 3, Item = "Fish tazer", Price = 200.0], [OrderID = 5, CustomerID = 3, Item = "Bandaids", Price = 2.0], [OrderID = 6, CustomerID = 1, Item = "Tackle box", Price = 20.0], [OrderID = 7, CustomerID = 5, Item = "Bait", Price = 3.25], [OrderID = 8, CustomerID = 5, Item = "Fishing Rod", Price = 100.0], [OrderID = 9, CustomerID = 6, Item = "Bait", Price = 3.25]}), {"OrderID"})
Table.FromRecords({[OrderID=1,CustomerID=1,Item="Fishing rod",Price=100],[OrderID=2,CustomerID=1,Item="1 lb. worms",Price=5],[OrderID=3,CustomerID=2,Item="Fishing net",Price=25],[OrderID=4,CustomerID=3,Item="Fish tazer",Price=200],[OrderID=5,CustomerID=3,Item="Bandaids",Price=2],[OrderID=6,CustomerID=1,Item="Tackle box",Price=20],[OrderID=7,CustomerID=5,Item="Bait",P...
```

● Table.SplitColumn

使用指定的拆分器功能，将指定的列拆分为一组其他列。

```
Table.SplitColumn( table as table, sourceColumn as text, splitter as function, optional columnNamesOrNumber as any, optional default as any, optional extraColumns as any) as table
```

```
将“i”处的 [Name] 列拆分为两列
let
Customers = Table.FromRecords({
[CustomerID = 1, Name = "Bob", Phone = "123-4567"],
[CustomerID = 2, Name = "Jim", Phone = "987-6543"],
[CustomerID = 3, Name = "Paul", Phone = "543-7890"],
[CustomerID = 4, Name = "Cristina", Phone = "232-1550"]
})
in
Table.SplitColumn(Customers,"Name",Splitter.SplitTextByDelimiter("i"),2)
Table.FromRecords({
[CustomerID = 1, Name.1 = "Bob", Name.2 = null, Phone = "123-4567"],
[CustomerID = 2, Name.1 = "J", Name.2 = "m", Phone = "987-6543"],
[CustomerID = 3, Name.1 = "Paul", Name.2 = null, Phone = "543-7890"],
[CustomerID = 4, Name.1 = "Cr", Name.2 = "st", Phone = "232-1550"]
})
```

Table.ToColumns

从表 table 中创建嵌套表的列表。 每个列表项都是一个包含列值的内部列表。

Table.ToColumns(table as table) as list

从表中创建列值的列表。

Table.ToColumns(Table.FromRecords({[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"] }))

{{1, 2},{"Bob", "Jim"},{ "123-4567", "987-6543"}}

Table.ToList

通过将指定的组合函数应用于表中的每一行值，将表转换为列表。

Table.ToList(table as table, optional combiner as nullable function) as list

使用逗号合并每行文本。

Table.ToList(Table.FromRows({{Number.ToText(1),"Bob", "123-4567" }, {Number.ToText(2), "Jim", "987-6543" }, {Number.ToText(3), "Paul", "543-7890" }}), Combiner.CombineTextByDelimiter(","))

{"1,Bob,123-4567", "2,Jim,987-6543", "3,Paul,543-7890"}

Table.ToRecords

将表 table 转换为记录列表。

Table.ToRecords(table as table) as list

将表转换为记录列表。

Table.ToRecords(Table.FromRows({{1, "Bob", "123-4567"} , {2, "Jim", "987-6543"}, {3, "Paul", "543-7890"} },{"CustomerID", "Name", "Phone"}))

{[CustomerID = 1, Name = "Bob", Phone = "123-4567"], [CustomerID = 2, Name = "Jim", Phone = "987-6543"], [CustomerID = 3, Name = "Paul", Phone = "543-7890"]}

Table.ToRows

从表 table 中创建嵌套表的列表。 每个列表项都是一个包含行值的内部列表。

Table.ToRows(table as table) as list

从表中创建行值的列表。

Table.ToRows(Table.FromRecords({[CustomerID =1, Name ="Bob", Phone = "123-4567"],[CustomerID =2, Name ="Jim", Phone = "987-6543"],[CustomerID =3, Name ="Paul", Phone = "543-7890"]}))

{{1, "Bob", "123-4567"},{2, "Jim", "987-6543"},{3, "Paul", "543-7890"}}

Table.TransformColumnNames

使用给定的 nameGenerator 函数转换列名。有效选项:

MaxLength 指定新列名的最大长度。如果给定函数生成的列名较长，则长名称将被剪裁。

Comparer 用于在生成新列名时控制比较。比较器可用于提供不区分大小写的比较或识别区域性与区域设置的比较。

公式语言中提供了以下内置比较器:

Comparer.Ordinal : 用于执行完全序号比较

Comparer.OrdinalIgnoreCase : 用于执行不区分大小写的完全序号比较

Comparer.FromCulture : 用于执行识别区域性的比较

Table.TransformColumnNames(table as table, nameGenerator as function, optional options as nullable record) as table

从列名中删除 #(tab) 字符

```
Table.TransformColumnNames(Table.FromRecords({[#"Col#(tab)umn" = 1]}), Text.Clean)
Table.FromRecords({[Column = 1]}, {"Column"})
```

转换列名以生成不区分大小写的名称(长度为 6)。

```
Table.TransformColumnNames(Table.FromRecords({[ColumnNum = 1, cOlumnnum = 2, coLumnNUM = 3]}), Text.Clean, [MaxLength = 6, Comparer = Comparer.OrdinalIgnoreCase])
Table.FromRecords({[Column = 1, cOlum1 = 2, coLum2 = 3]}, {"Column", "cOlum1", "coLum2"})
```

● Table.TransformColumnTypes

通过对在参数 typeTransformations 中指定的列应用转换操作(其中格式为 { column name, type name})，使用参数 culture 中的指定区域性，从输入 table 中返回一个表。

如果该列不存在，则引发异常。

Table.TransformColumnTypes(table as table, typeTransformations as list, optional culture as nullable text) as table

在表 ({[a = 1, b = 2], [a = 3, b = 4]}) 中将列 [a] 中的数值转换为文本值。

```
Table.TransformColumnTypes(Table.FromRecords({[a = 1, b = 2], [a = 3, b = 4]}), {"a", type text}, "en-US")
Table.FromRecords({[a = "1", b = 2],
[a = "3", b = 4]})
```

● Table.TransformColumns

通过对在参数 transformOperations 中指定的列应用转换操作(其中格式为 { column name, transformation })，从输入 table 中返回一个表。

如果此列不存在，将引发异常，除非可选参数 defaultTransformation 指定备用值(例如，MissingField.UseNull 或 MissingField.Ignore)。

```
Table.TransformColumns( table as table, transformOperations as list, optional defaultTransformation as nullable function, optional missingField as nullable number) as table
```

将列 [A] 中的数字值转换为数字值。

```
Table.TransformColumns(Table.FromRecords({[A="1", B=2], [A="5", B=10]}),{"A", Number.FromText})
Table.FromRecords({[A=1, B=2], [A=5, B=10]})
----------
```

将缺失列 [X] 中的数字值转换为文本值，同时忽略不存在的列。

```
Table.TransformColumns(Table.FromRecords({[A="1", B=2], [A="5", B=10]}), {"X", Number.FromText}, null, MissingField.Ignore)
Table.FromRecords({[A="1", B=2], [A="5", B=10]})
----------
```

将缺失列 [X] 中的数字值转换为文本值，同时将不存在的列中的值默认设置为 null。

```
Table.TransformColumns(Table.FromRecords({[A="1",B=2], [A="5", B=10]}), {"X", Number.FromText}, null, MissingField.UseNull)
Table.FromRecords({[A="1", B=2, X=null], [A="5", B=10, X=null]})
----------
```

将缺失列 [X] 中的数字值转换为文本值，其中不存在的列出错。

```
Table.TransformColumns(Table.FromRecords({[A="1",B=2], [A="5", B=10]}), {"X", Number.FromText})
[Expression.Error] The column 'X' of the table wasn't found.
```

● Table.TransformRows

通过将 transform 操作应用于行，从 table 创建一个表。

如果指定了 transform 函数的返回类型，则结果将是具有该行类型的表。 在所有其他情况下，此函数的结果将是一个列表，其中的项类型为转换函数的返回类型。

```
Table.TransformRows( table as table, transform as function) as list
```

在表 ({[A = 1], [A = 2], [A = 3], [A = 4], [A = 5]}) 中将行转换为数值列表。

```
Table.TransformRows(Table.FromRecords({[a = 1], [a = 2], [a = 3], [a = 4], [a = 5]}), each [a])
{1, 2, 3, 4, 5}
----------
```

在表 ({[A = 1], [A = 2], [A = 3], [A = 4], [A = 5]) 中将列 [A] 中的行转换为列 [B] 中的文本值。

```
Table.TransformRows(Table.FromRecords({[a = 1], [a = 2], [a = 3], [a = 4], [a = 5]}), (row) as record => [B = Number.ToText(row[a])])
{ [
B = "1"
], [
B = "2"
], [
B = "3"
], [
B = "4"
], [
B = "5"
] }
```

● Table.Transpose

使列成为行，并使行成为列。

```
Table.Transpose( table as table, optional columns as any) as table
```

使名称-值对的表行成为列。

```
Table.Transpose(Table.FromRecords({[Name = "Full Name", Value = "Fred"], [Name = "Age", Value = 42], [Name = "Country", Value = "UK"]}))
Table.FromRecords({ [
Column1 = "Full Name",
Column2 = "Age",
Column3 = "Country"
], [
Column1 = "Fred",
Column2 = 42,
Column3 = "UK"
```

```
]
}, {
"Column1",
"Column2",
"Column3"
})
```

● Table.Type

表示所有表的类型。

● Table.Unpivot

将表中的一组列转换为属性-值对，并与每行中的剩余值相结合。

```
Table.Unpivot( table as table, pivotColumns as list, attributeColumn as text, valueColumn as text) as table
```

```
选取表 ({[ key = "x", a = 1, b = null, c = 3 ], [ key = "y", a = 2, b = 4, c = null ]}) 中的列 "a"、"b" 和 "c"，并将它们逆透视为属性-值对。

Table.Unpivot(Table.FromRecords({[ key = "x", a = 1, b = null, c = 3 ], [ key = "y", a = 2, b = 4, c = null ]}), { "a", "b", "c" }, "attribute", "value")
Table.FromRecords({ [ key = "x", attribute = "a", value = 1 ], [ key = "x", attribute = "c", value = 3 ], [ key = "y", attribute = "a", value = 2 ], [ key = "y", attribute = "b", value = 4 ] })
```

● Table.UnpivotOtherColumns

将指定集以外的所有列转换为属性值对，并与每行中的剩余值合并。

```
Table.UnpivotOtherColumns( table as table, pivotColumns as list, attributeColumn as text, valueColumn as text) as table
```

```
将指定集以外的所有列转换为属性值对，并与每行中的剩余值合并。

Table.UnpivotOtherColumns(Table.FromRecords({
[ key = "key1", attribute1 = 1, attribute2 = 2, attribute3 = 3 ],
[ key = "key2", attribute1 = 4, attribute2 = 5, attribute3 = 6 ]
}), { "key" }, "column1", "column2")
Table.FromRecords({
[ key = "key1", column1 = "attribute1", column2 = 1 ],
[ key = "key1", column1 = "attribute2", column2 = 2 ],
[ key = "key1", column1 = "attribute3", column2 = 3 ],
[ key = "key2", column1 = "attribute1", column2 = 4 ],
[ key = "key2", column1 = "attribute2", column2 = 5 ],
[ key = "key2", column1 = "attribute3", column2 = 6 ]
})
```

● Table.View

返回 table 的视图，其中，如果将操作应用到视图，则使用 handlers 中指定的函数替代操作的默认行为。

处理程序函数是可选的。如果没有为操作指定处理程序函数，则将操作的默认行为应用到 table （GetExpression 除外)。

处理程序函数返回的值必须在语义上等同于针对 table 应用操作的结果(如果是 GetExpression ，则为结果视图)。

如果处理程序函数引发一个错误，则将操作的默认行为应用到视图。

Table.View 可用于实现数据源的折叠 - 将 M 查询转换为特定于源的查询(例如，从 M 查询创建 T-SQL 语句)。

有关 Table.View 的更多完整说明，请查看已发布文档。

Table.View(table as nullable table, handlers as record) as table

● Tables.GetRelationships

获取一组表之间的关系。假定集 tables 的结构与导航表的结构相似。 dataColumn 定义的列包含实际数据表。

Tables.GetRelationships(tables as table, optional dataColumn as nullable text) as table

● Teradata.Database

从服务器 server 上的 Teradata 数据库返回包含 SQL 表和视图的表。可以视需要使用服务器指定端口，并用冒号分隔。可以指定可选的记录参数 options 来控制以下选项:

CreateNavigationProperties :逻辑(true/false)，用于设置是否在返回的值上生成导航属性(默认值为 true)。

NavigationPropertyNameGenerator :函数，用于创建导航属性的名称。

Query :文本，用于转换为在服务器上运行的 SQL 查询。你可以指定多个查询，但系统只返回第一个结果。

CommandTimeout :持续时间，用于控制服务器端查询在取消前运行的时间。默认值为十分钟。

HierarchicalNavigation :逻辑(true/false)，用于设置是否查看按架构名称分组的表(默认值为false)。

例如，可以将记录参数指定为 [option1 = value1, option2 = value2...] 或 [Query = "select ..."]。

```
Teradata.Database( server   as text, optional   options   as nullable record) as table
```

- **Text.At**

在位置 index 返回文本值中的字符 text 。文本中的第一个字符位于位置 0。

```
Text.At( text   as nullable text,   index   as number) as nullable text
```

```
查找位于字符串 "Hello, World" 中位置 4 的字符。

Text.At("Hello, World", 4)
"o"
```

- **Text.Clean**

返回 text 的所有非打印字符均已删除的文本值。

```
Text.Clean( text   as nullable text) as nullable text
```

```
从文本值中删除换行和其他非打印字符。

Text.Clean("ABC#(lf)D")
"ABCD"
```

- **Text.Combine**

返回将一系列文本值 texts 组合为单个文本值的结果。

可以指定最终组合文本中使用的可选分隔符 separator 。

```
Text.Combine( texts   as list, optional   separator   as nullable text) as text
```

```
组合文本值 "Seattle" 和 "WA"。
```

Text.Combine({"Seattle", "WA"})
"SeattleWA"

组合文本值 "Seattle" 和 "WA"，以逗号和空格 ", " 分隔。

Text.Combine({"Seattle", "WA"}, ", ")
"Seattle, WA"

● Text.Contains

检测文本 text 是否包含文本 substring 。如果找到此文本，则返回 true。

comparer 是用于控制比较的 Comparer 。比较器可用于提供不区分大小写的比较或识别区域性与区域设置的比较。

公式语言中提供了以下内置比较器:

Comparer.Ordinal : 用于执行完全序号比较

Comparer.OrdinalIgnoreCase : 用于执行不区分大小写的完全序号比较

Comparer.FromCulture : 用于执行识别区域性的比较

Text.Contains(text as nullable text, substring as text, optional comparer as nullable function) as nullable logical

查找文本 "Hello World" 是否包含 "Hello"。

Text.Contains("Hello World", "Hello")
true

查找文本 "Hello World" 是否包含 "hello"。

Text.Contains("Hello World", "hello")
false

● Text.End

返回一个 text 值，该值是 text 值 text 的后 count 个字符。

Text.End(text as nullable text, count as number) as nullable text

获取文本 "Hello, World" 的后 5 个字符。

Text.End("Hello, World", 5)
"World"

- **Text.EndsWith**

指示给定的文本 text 是否以指定的值 substring 结尾。指示内容区分大小写。

comparer 是用于控制比较的 Comparer 。比较器可用于提供不区分大小写的比较或识别区域性与区域设置的比较。

公式语言中提供了以下内置比较器:

Comparer.Ordinal : 用于执行完全序号比较

Comparer.OrdinalIgnoreCase : 用于执行不区分大小写的完全序号比较

Comparer.FromCulture : 用于执行识别区域的比较

Text.EndsWith(text as nullable text, substring as text, optional comparer as nullable function) as nullable logical

检查 "Hello, World" 是否以 "world" 结尾。

Text.EndsWith("Hello, World", "world")
false

检查 "Hello, World" 是否以 "World" 结尾。

Text.EndsWith("Hello, World", "World")
true

- **Text.Format**

返回通过将来自列表或记录的 arguments 应用于格式字符串 formatString 创建的格式化文本。可以视情况指定区域性。

Text.Format(formatString as text, arguments as any, optional culture as nullable text) as text

设置数字列表格式。

Text.Format("#{0}, #{1}, and #{2}.", { 17, 7, 22 })
"17, 7, and 22."

根据美国英语区域性设置记录中的不同数据类型的格式。

Text.Format("The time for the #[distance] km run held in #[city] on #[date] was #[duration].", [city = "Seattle", date = #date(2015, 3, 10), duration = #duration(0,0,54,40), distance = 10], "en-US")
"The time for the 10 km run held in Seattle on 3/10/2015 was 00:54:40."

● Text.From

返回 value 的文本表示形式。 value 可以是 number 、 date 、 time 、 datetime 、 datetimezone 、 logical 、 duration 或 binary 值。

如果给定值为 Null， Text.From 将返回 Null。还可以提供可选的 culture 。

Text.From(value as any, optional culture as nullable text) as nullable text

从数字 3 创建一个文本值。

Text.From(3)
"3"

● Text.FromBinary

使用 encoding 类型将数据 binary 从二进制值解码为文本值。

Text.FromBinary(binary as nullable binary, optional encoding as nullable number) as nullable text

● Text.Insert

返回将文本值 newText 插入到文本值 text 中的位置 offset 的结果。位置从数字 0 开始。

Text.Insert(text as nullable text, offset as number, newText as text) as nullable text

在 "ABD" 中的 "B" 和 "D" 之间插入 "C"。

Text.Insert("ABD", 2, "C")
"ABCD"

● Text.Length

返回文本 text 中的字符数。

Text.Length(text as nullable text) as nullable number

查找文本 "Hello World" 中有多少个字符。

```
Text.Length("Hello World")
11
```

Text.Lower

返回将 text 中的所有字符转换为小写的结果。

```
Text.Lower( text as nullable text, optional culture as nullable text) as nullable text
```

```
获取 "AbCd" 的小写版本。

Text.Lower("AbCd")
"abcd"
```

Text.Middle

返回 count 个字符，或返回至 text 的结束；采用偏移 start 。

```
Text.Middle( text as nullable text, start as number, optional count as nullable number) as nullable text
```

```
从文本 "Hello World" 中查找从索引 6 开始、跨 5 个字符的子字符串。

Text.Middle("Hello World", 6, 5)
"World"
----------

从文本 "Hello World" 中查找从索引 6 开始到结束的子字符串。

Text.Middle("Hello World", 6, 20)
"World"
```

Text.NewGuid

返回新的、随机的全局唯一标识符(GUID)。

```
Text.NewGuid() as text
```

Text.PadEnd

通过在文本值 text 的尾部插入空格，返回填充到长度 count 的 text 值。

可选字符 character 可用于指定用于填充的字符。默认的填充字符是空格。

```
Text.PadEnd( text as nullable text, count as number, optional character as nullable text) as nullable text
```

```
填充文本值的尾部，使其长度为 10 个字符。

Text.PadEnd("Name", 10)
"Name "
----------
```

用 "|" 填充文本值的尾部，使其长度为 10 个字符。

```
Text.PadEnd("Name", 10, "|")
"Name||||||"
```

- ## Text.PadStart

通过在文本值 text 的开头插入空格，返回填充到长度 count 的 text 值。

可选字符 character 可用于指定用于填充的字符。默认的填充字符是空格。

Text.PadStart(text as nullable text, count as number, optional character as nullable text) as nullable text

填充文本值的开头，使其长度为 10 个字符。

```
Text.PadStart("Name", 10)
" Name"
----------
```

用 "|" 填充文本值的开头，使其长度为 10 个字符。

```
Text.PadStart("Name", 10, "|")
"||||||Name"
```

- ## Text.PositionOf

返回在 text 中找到的文本值 substring 指定出现次数的位置。

可选参数 occurrence 可用于指定要返回的出现位置(默认值为第一次出现的位置)。

comparer 为用于控制比较的 Comparer 。比较器可用于提供不区分大小写的比较或识别区域性与区域设置的比较。

公式语言中提供了以下内置比较器:

Comparer.Ordinal : 用于执行完全序号比较

Comparer.OrdinalIgnoreCase : 用于执行不区分大小写的完全序号比较

Comparer.FromCulture：用于执行识别区域性的比较

Text.PositionOf(text as text, substring as text, optional occurrence as nullable number, optional comparer as nullable function) as any

获取 "World" 在文本 "Hello, World! Hello, World!" 中第一次出现的位置。

```
Text.PositionOf("Hello, World! Hello, World!", "World")
7
----------
```

获取 "World" 在 "Hello, World! Hello, World!" 中最后一次出现的位置。

```
Text.PositionOf("Hello, World! Hello, World!", "World", Occurrence.Last)
21
```

● Text.PositionOfAny

返回在文本值 characters 中找到的字符列表 text 中的任何字符第一次出现的位置。

可选参数 occurrence 可用于指定要返回的出现位置。

Text.PositionOfAny(text as text, characters as list, optional occurrence as nullable number) as any

查找 "W" 在文本 "Hello, World!" 中的位置。

```
Text.PositionOfAny("Hello, World!", {"W"})
7
----------
```

查找 "W" 或 "H" 在文本 "Hello, World!" 中的位置。

```
Text.PositionOfAny("Hello, World!", {"H","W"})
0
```

● Text.Proper

返回只使文本值 text 中每个字词的第一个字符大写的结果。所有其他字母均以小写返回。

Text.Proper(text as nullable text, optional culture as nullable text) as nullable text

对简单句子使用 Text.Proper。

```
Text.Proper("the QUICK BrOWn fOx jUmPs oVER tHe LAzy DoG")
"The Quick Brown Fox Jumps Over The Lazy Dog"
```

● Text.Range

从文本 text 中返回在偏移量 offset 中找到的子字符串。

可以包含一个可选参数 count ，以指定要返回多少个字符。

Text.Range(text as nullable text, offset as number, optional count as nullable number) as nullable text

从文本 "Hello World" 中查找从索引 6 开始的子字符串。

```
Text.Range("Hello World", 6)
"World"
```

从文本 "Hello World Hello" 中查找从索引 6 开始且涵盖 5 个字符的子字符串。

```
Text.Range("Hello World Hello", 6, 5)
"World"
```

● Text.Remove

返回文本值 text 已删除了 removeChars 的所有字符的副本。

Text.Remove(text as nullable text, removeChars as any) as nullable text

从文本值中删除字符 ， 和 ;。

```
Text.Remove("a,b;c",{",",";"})
"abc"
```

● Text.RemoveRange

返回文本值 text 已删除了位置 offset 的所有字符的副本。

可选参数 count 可用于指定要删除的字符数。 count 的默认值为 1。位置值从 0 开始。

Text.RemoveRange(text as nullable text, offset as number, optional count as nullable number) as nullable text

从文本值 "ABEFC" 的位置 2 删除 1 个字符。

```
Text.RemoveRange("ABEFC", 2)
"ABFC"
```

从文本值 "ABEFC" 中删除从位置 2 开始的两个字符。

```
Text.RemoveRange("ABEFC", 2, 2)
"ABC"
```

● Text.Repeat

返回由输入文本 text 重复 count 次而组成的文本值。

Text.Repeat(text as nullable text, count as number) as nullable text

重复文本 "a" 五次。

```
Text.Repeat("a", 5)
"aaaaa"
```

重复文本 "helloworld" 三次。

```
Text.Repeat("helloworld.", 3)
"helloworld.helloworld.helloworld."
```

● Text.Replace

返回将文本值 text 中所有出现的文本值 old 替换为文本值 new 的结果。此函数区分大小写。

```
Text.Replace( text as nullable text, old as text, new as text) as nullable text
```

将句子中出现的每个 "the" 替换为 "a"。

```
Text.Replace("the quick brown fox jumps over the lazy dog", "the", "a")
"a quick brown fox jumps over a lazy dog"
```

● Text.ReplaceRange

返回从文本值 text 中的位置 offset 开始删除一些字符 count ，然后在 text 中的相同位置插入文本值 newText 的结果。

```
Text.ReplaceRange( text as nullable text, offset as number, count as number, newText as text) as nullable text
```

使用新文本值 "CDE" 替换文本值 "ABGF" 中位置 2 的单个字符。

```
Text.ReplaceRange("ABGF", 2, 1, "CDE")
"ABCDEF"
```

● Text.Split

返回根据指定的分隔符 separator 拆分文本值 text 而得到的文本值列表。

```
Text.Split( text as text, separator as text) as list
```

从由 "|" 分隔的文本值 "Name|Address|PhoneNumber" 创建列表。

```
Text.Split("Name|Address|PhoneNumber", "|")
{
"Name",
"Address",
"PhoneNumber"
```

```
}
```

- **Text.SplitAny**

返回根据指定的分隔符 separators 中的任意字符拆分文本值 text 而得到的文本值列表。

```
Text.SplitAny( text as text, separators as text) as list

    从文本值 "Jamie|Campbell|Admin|Adventure Works|www.adventure-works.com" 创建列表。

    Text.SplitAny("Jamie|Campbell|Admin|Adventure Works|www.adventure-works.com", "|")
    {"Jamie",
    "Campbell",
    "Admin",
    "Adventure Works",
    "www.adventure-works.com"}
```

- **Text.Start**

返回 text 的前 count 个字符作为文本值。

```
Text.Start( text as nullable text, count as number) as nullable text

    获取 "Hello, World" 的前 5 个字符。

    Text.Start("Hello, World", 5)
    "Hello"
```

- **Text.StartsWith**

如果文本值 text 以文本值 substring 开头，则返回 true。

text : 要搜索的 text 值

substring : 一个 text 值，该值是要在 substring 中进行搜索的子字符串

comparer : [可选] 用于控制比较的 Comparer 。例如， Comparer.OrdinalIgnoreCase 可用于执行不区分大小写的搜索

comparer 是用于控制比较的 Comparer 。比较器可用于提供不区分大小写的比较或识别区域性与区域设置的比较。

公式语言中提供了以下内置比较器:

Comparer.Ordinal：用于执行完全序号比较

Comparer.OrdinalIgnoreCase：用于执行不区分大小写的完全序号比较

Comparer.FromCulture：用于执行识别区域性的比较

Text.StartsWith(text as nullable text, substring as text, optional comparer as nullable function) as nullable logical

检查文本 "Hello, World" 是否以文本 "hello" 开头。

```
Text.StartsWith("Hello, World", "hello")
false
----------
```

检查文本 "Hello, World" 是否以文本 "Hello" 开头。

```
Text.StartsWith("Hello, World", "Hello")
true
```

● Text.ToBinary

使用指定的 encoding 将给定的文本值 text 编码为二进制值。

Text.ToBinary(text as nullable text, optional encoding as nullable number, optional includeByteOrderMark as nullable logical) as nullable binary

● Text.ToList

从给定的文本值 text 返回字符值列表。

Text.ToList(text as text) as list

从文本 "Hello World" 创建字符值列表。

```
Text.ToList("Hello World")
{"H",
"e",
"l",
"l",
"o",
" ",
"W",
"o",
"r",
"l",
"d"}
```

Text.Trim

返回从文本值 text 删除所有前导空格和尾随空格的结果。

```
Text.Trim( text as nullable text, optional trim as any) as nullable text
```

从 " a b c d " 删除前导空格和尾随空格。

```
Text.Trim(" a b c d ")
"a b c d"
```

Text.TrimEnd

返回从文本值 text 删除所有尾随空格的结果。

```
Text.TrimEnd( text as nullable text, optional trim as any) as nullable text
```

从 " a b c d " 删除尾随空格。

```
Text.TrimEnd(" a b c d ")
" a b c d"
```

Text.TrimStart

返回从文本值 text 删除所有前导空格的结果。

```
Text.TrimStart( text as nullable text, optional trim as any) as nullable text
```

从 " a b c d " 删除前导空格。

```
Text.TrimStart(" a b c d ")
"a b c d "
```

Text.Type

表示所有文本值的类型。

Text.Upper

返回将 text 中的所有字符转换为大写的结果。

```
Text.Upper( text as nullable text, optional culture as nullable text) as nullable text
```

获取 "aBcD" 的大写版本。

```
Text.Upper("aBcD")
"ABCD"
```

TextEncoding.Ascii

用于选择 ASCII 二进制格式。

- **TextEncoding.BigEndianUnicode**

用于选择 UTF16 big endian 二进制格式。

- **TextEncoding.Type**

指定文本编码类型。

- **TextEncoding.Unicode**

用于选择 UTF16 little endian 二进制格式。

- **TextEncoding.Utf16**

用于选择 UTF16 little endian 二进制格式。

- **TextEncoding.Utf8**

用于选择 UTF8 二进制格式。

- **TextEncoding.Windows**

用于选择 Windows 二进制格式。

- **Time.EndOfHour**

返回表示 dateTime 中的小时结尾的 time 、 datetime 或 datetimezone 值，包括秒的小数形式。保留时区信息。

dateTime：从中计算小时结尾的 time 、 datetime 或 datetimezone 值。

Time.EndOfHour(dateTime as any) as any

获取 5/14/2011 05:00:00 PM 的小时结尾。

```
Time.EndOfHour(#datetime(2011, 5, 14, 17, 0, 0))
#datetime(2011, 5, 14, 17, 59, 59.9999999)
```

获取 5/17/2011 05:00:00 PM -7:00 的小时结尾。

```
Time.EndOfHour(#datetimezone(2011, 5, 17, 5, 0, 0, -7, 0))
#datetimezone(2011, 5, 17, 5, 59, 59.9999999, -7, 0)
```

- **Time.From**

从给定的 value 返回 time 值。如果给定的 value 是 null ， Time.From 将返回 null 。 如果给定的 value 是 time ，则返回 value 。以下类型的值可以转换为 time 值:

text : 文本表示形式的 time 值。有关详细信息，请参阅 Time.FromText 。

datetime : value 的时间部分。

datetimezone : 本地日期时间中等效于 value 的时间部分。

number : 与由 value 表示的不完整天数等效的 time 。如果 value 是负数或大于等于 1，则返回错误。

如果 value 属于任何其他类型，则返回错误。

```
Time.From( value  as any, optional  culture  as nullable text) as nullable time
```

```
将 0.7575 转换为 time 值。

Time.From(0.7575)
#time(18,10,48)
----------

将 #datetime(1899, 12, 30, 06, 45, 12) 转换为 time 值。

Time.From(#datetime(1899, 12, 30, 06, 45, 12))
#time(06, 45, 12)
```

● Time.FromText

按照 ISO 8601 格式标准，从文本表示形式 text 创建 time 值。

```
Time.FromText("12:34:12")    // 时间，hh:mm:ss
Time.FromText("12:34:12.1254425")    // hh:mm:ss.nnnnnnn
```

```
Time.FromText( text  as nullable text, optional  culture  as nullable text) as nullable time
```

```
将 "10:12:31am" 转换为时间值。

Time.FromText("10:12:31am")
#time(10, 12, 31)
----------

将 "1012" 转换为 Time 值。

Time.FromText("1012")
```

```
#time(10, 12, 00)
----------------
```

将 "10" 转换为 Time 值。

```
Time.FromText("10")
#time(10, 00, 00)
```

Time.Hour

返回所提供的 time 、 datetime 或 datetimezone 值 dateTime 的小时部分。

```
Time.Hour( dateTime  as any) as nullable number
```

查找 #datetime(2011, 12, 31, 9, 15, 36) 中的小时。

```
Time.Hour(#datetime(2011, 12, 31, 9, 15, 36))
9
```

Time.Minute

返回所提供的 time 、 datetime 或 datetimezone 值 dateTime 的分钟部分。

```
Time.Minute( dateTime  as any) as nullable number
```

查找 #datetime(2011, 12, 31, 9, 15, 36) 中的分钟。

```
Time.Minute(#datetime(2011, 12, 31, 9, 15, 36))
15
```

Time.Second

返回所提供的 time 、 datetime 或 datetimezone 值 dateTime 的秒部分。

```
Time.Second( dateTime  as any) as nullable number
```

查找日期时间值中的秒值。

```
Time.Second(#datetime(2011, 12, 31, 9, 15, 36.5))
36.5
```

Time.StartOfHour

返回给定 time 、 datetime 或 datetimezone 类型的第一个小时值。

```
Time.StartOfHour( dateTime  as any) as any
```

查找 2011 年 10 月 10 日上午 8:10:32 (#datetime(2011, 10, 10, 8, 10, 32)) 的小时开头值。

```
Time.StartOfHour(#datetime(2011, 10, 10, 8, 10, 32))
#datetime(2011, 10, 10, 8, 0, 0)
```

Time.ToRecord

返回包含给定时间值 time 的各个部分的记录。

time：要从中计算其各个部分的记录的 time 值。

```
Time.ToRecord( time  as time) as record
```

将 #time(11, 56, 2) 值转换为包含时间值的记录。

```
Time.ToRecord(#time(11, 56, 2))
[
Hour = 11,
Minute = 56,
Second = 2
]
```

● Time.ToText

返回 time 时间值的文本表示形式 time 。

此函数采用一个可选格式参数 format 。有关支持的格式的完整列表，请参阅库规范文档。

```
Time.ToText( time  as nullable time, optional  format  as nullable text, optional  culture  as nullable text) as nullable text
```

获取 #time(11，56，2) 的文本表示形式。

```
Time.ToText(#time(11, 56, 2))
"11:56 AM"
----------
```

使用格式选项获取 #time(11, 56, 2)的文本表示形式。

```
Time.ToText(#time(11, 56, 2), "hh:mm")
"11:56"
```

● Time.Type

表示所有时间值的类型。

● TraceLevel.Critical

返回严重跟踪级别值 1。

● TraceLevel.Error

返回错误跟踪级别值 2。

● TraceLevel.Information

返回信息跟踪级别值 4。

- **TraceLevel.Type**

指定跟踪级别。

- **TraceLevel.Verbose**

返回详细跟踪级别值 5。

- **TraceLevel.Warning**

返回警告跟踪级别值 3。

- **Type.AddTableKey**

向给定表类型添加键。

```
Type.AddTableKey( table  as type,  columns  as list,  isPrimary  as logical) as type
```

- **Type.ClosedRecord**

返回给定 record type (或同一类型，如果其已关闭)的已关闭版本。

```
Type.ClosedRecord( type  as type) as type
```

```
创建 type [ A = number,…] 的已关闭版本。
```

```
Type.ClosedRecord(type [ A = number,...])
type [
A = number
]
```

- **Type.Facets**

返回一条包含 type 的 Facet 的记录。

```
Type.Facets( type  as type) as record
```

- **Type.ForFunction**

从 signature 、 ReturnType 和 Parameters 、 min 以及调用函数所需参数的最小数目中创建 function type 。

```
Type.ForFunction( signature  as record,  min  as number) as type
```

```
决定记录 type [ A = number, ...] 是否打开。
```

```
Type.ForFunction([ReturnType = type number, Parameters = [X = type number]], 1)
type function (X as number) as number
```

Type.ForRecord

返回一个类型，此类型表示对字段具有特定类型约束的记录。

```
Type.ForRecord( fields  as record,  open  as logical) as type
```

Type.FunctionParameters

返回带有字段值的记录以设置 type 的参数名称，其值设置为对应类型。

```
Type.FunctionParameters( type  as type) as record
```

```
找到函数 (x as number, y as text) 的参数类型。
```

```
Type.FunctionParameters(type function (x as number, y as text) as any)
[x = type number, y = type text]
```

Type.FunctionRequiredParameters

返回一个数字，表明调用函数的输入 type 所需参数的最小数量。

```
Type.FunctionRequiredParameters( type  as type) as number
```

```
找到函数 (x as number, optional y as text) 所需参数的数量。
```

```
Type.FunctionRequiredParameters(type function (x as number, optional y as text) as any)
1
```

Type.FunctionReturn

返回由函数 type 返回的类型。

```
Type.FunctionReturn( type  as type) as type
```

```
找到 () as any) 的返回类型。
```

```
Type.FunctionReturn(type function () as any)
type any
```

Type.Is

Type.Is

```
Type.Is( type1  as type,  type2  as type) as logical
```

Type.IsNullable

如果类型是 nullable 类型则返回 true ；否则，返回 false 。

```
Type.IsNullable( type  as type) as logical
```

确定 number 是否可空。

```
Type.IsNullable(type number)
false
----------
```

确定 type nullable number 是否可空。

```
Type.IsNullable(type nullable number)
true
```

Type.IsOpenRecord

返回 logical 表明记录 type 是否打开。

```
Type.IsOpenRecord( type as type) as logical
```

决定记录 type [A = number, ...] 是否打开。

```
Type.IsOpenRecord(type [ A = number,...])
true
```

Type.ListItem

从列表 type 中返回项类型。

```
Type.ListItem( type as type) as type
```

从列表 {number} 中找到项类型。

```
Type.ListItem(type {number})
type number
```

Type.NonNullable

从 type 返回非 nullable 类型。

```
Type.NonNullable( type as type) as type
```

返回 type nullable number 的非可空类型。

```
Type.NonNullable(type nullable number)
type number
```

Type.OpenRecord

返回给定 record type (或同一类型，如果其已打开)的打开版本。

```
Type.OpenRecord( type as type) as type
```

创建 type [A = number] 的打开版本。

```
Type.OpenRecord(type [ A = number])
```

```
type [
A = number, ...
]
```

Type.RecordFields

返回描述记录 type 的字段的记录。在记录 [Type = type, Optional = logical] 的格式中，返回的记录类型的每个字段都有对应的名称和值。

```
Type.RecordFields( type as type) as record
```

找到记录 [A = number, optional B = any] 的名称和值。

```
Type.RecordFields(type [ A = number, optional B = any])
[ A = [Type = type number, Optional = false], B = [Type = type any, Optional = true] ]
```

Type.ReplaceFacets

将 type 的 Facet 替换为记录 facets 中包含的 Facet。

```
Type.ReplaceFacets( type as type, facets as record) as type
```

Type.ReplaceTableKeys

返回所有键均由指定的键列表替换的新的表类型。

```
Type.ReplaceTableKeys( tableType as type, keys as list) as type
```

Type.TableColumn

返回表类型 tableType 中列 column 的类型。

```
Type.TableColumn( tableType as type, column as text) as type
```

Type.TableKeys

返回给定表类型的可能为空的键列表。

```
Type.TableKeys( tableType as type) as list
```

Type.TableRow

Type.TableRow

```
Type.TableRow( table as type) as type
```

Type.TableSchema

返回说明 tableType 的列的表。

请参阅有关 Table.Schema 的文档，了解结果表的说明。

```
Type.TableSchema( tableType  as type) as table
```

- **Type.Type**

表示所有类型的类型。

- **Type.Union**

返回 types 中类型的并集。

```
Type.Union( types  as list) as type
```

- **Uri.BuildQueryString**

将记录 query 汇编入 URI 查询字符串，根据需要转义字符。

```
Uri.BuildQueryString( query  as record) as text
```

```
对包含某些特殊字符的查询字符串进行编码。

Uri.BuildQueryString([a="1", b="+$"])
"a=1&b=%2B%24"
```

- **Uri.Combine**

返回一个绝对 URI，这是输入 baseUri 和 relativeUri 的组合。

```
Uri.Combine( baseUri  as text,  relativeUri  as text) as text
```

- **Uri.EscapeDataString**

根据 RFC 3986 的规则对输入 data 中的特殊字符进行编码。

```
Uri.EscapeDataString( data  as text) as text
```

```
对“+money$”中的特殊字符进行编码。

Uri.EscapeDataString("+money$")
"%2Bmoney%24"
```

- **Uri.Parts**

返回输入 absoluteUri 的组成部分作为记录，包含诸如方案、主机、端口、路径、查询、片段、用户名和密码等此类值。

```
Uri.Parts( absoluteUri  as text) as record
```

```
查找绝对 URI "www.adventure-works.com" 的组成部分。

Uri.Parts("www.adventure-works.com")
[ Scheme = "http",
```

```
Host = "www.adventure-works.com",
Port = 80,
Path = "/",
Query = [],
Fragment = "",
UserName = "",
Password = "" ]
----------------
```

解码百分比编码字符串。

```
let
UriUnescapeDataString = (data as text) as text => Uri.Parts("http://contoso?a=" & data)[Query][a]
in
UriUnescapeDataString("%2Bmoney%24")
"+money$"
```

- ## Value.Add

返回 value1 和 value2 的总和。可以指定一个可选 precision 参数，默认情况下使用 Precision.Double 。

```
Value.Add( value1 as any, value2 as any, optional precision as nullable number) as any
```

- ## Value.As

Value.As

```
Value.As( value as any, type as type) as any
```

- ## Value.Compare

根据第一个值是小于、等于还是大于第二个值，返回 -1、0 或 1。

```
Value.Compare( value1 as any, value2 as any, optional precision as nullable number) as number
```

- ## Value.Divide

返回将 value1 除以 value2 的结果。可以指定一个可选 precision 参数，默认情况下使用 Precision.Double 。

```
Value.Divide( value1 as any, value2 as any, optional precision as nullable number) as any
```

- ## Value.Equals

如果值 value1 等于值 value2 ，则返回 true，否则返回 false。

```
Value.Equals( value1 as any, value2 as any, optional precision as nullable number) as logical
```

Value.Firewall

Value.Firewall

Value.Firewall(key as text) as any

Value.FromText

从文本表示形式 text 对值解码，并将其解释为具有适当类型的值。

Value.FromText 采用一个文本值并返回数值、逻辑值、Null 值、日期时间值、期间值或文本值。 空文本值会被解释为 Null 值。

Value.FromText(text as any, optional culture as nullable text) as any

Value.Is

Value.Is

Value.Is(value as any, type as type) as logical

Value.Metadata

返回包含输入的元数据的记录。

Value.Metadata(value as any) as any

Value.Multiply

返回将 value1 与 value2 相乘的乘积。可以指定一个可选 precision 参数，默认情况下使用 Precision.Double 。

Value.Multiply(value1 as any, value2 as any, optional precision as nullable number) as any

Value.NativeQuery

使用 parameters 中指定的参数和 options 中指定的选项对 target 计算 query 。

target 定义查询的输出。

target 提供 query 所述运算的上下文。

query 描述了要对 target 执行的查询。 query 以 target 的专用方式(例如 T-SQL 语句)进行表示。

可选的 parameters 值可能包含适用于提供 query 预期的参数值的列表或记录。

可选的 options 记录可能包含影响 query 对 target 的计算行为的选项。这些选项为 target 所特有。

```
Value.NativeQuery( target as any, query as text, optional parameters as any, optional options as nullable record) as any
```

- **Value.NullableEquals**

如果任一参数“ value1 ”或“ value2 ”为 Null，则返回 Null，否则等同于 Value.Equals。

```
Value.NullableEquals( value1 as any, value2 as any, optional precision as nullable number) as nullable logical
```

- **Value.RemoveMetadata**

提取元数据的输入。

```
Value.RemoveMetadata( value as any, optional metaValue as any) as any
```

- **Value.ReplaceMetadata**

替换输入的元数据信息。

```
Value.ReplaceMetadata( value as any, metaValue as any) as any
```

- **Value.ReplaceType**

Value.ReplaceType

```
Value.ReplaceType( value as any, type as type) as any
```

- **Value.ResourceExpression**

Value.ResourceExpression

```
Value.ResourceExpression( value as any) as any
```

- **Value.Subtract**

返回 value1 和 value2 的差。可以指定一个可选 precision 参数，默认情况下使用 Precision.Double 。

```
Value.Subtract( value1 as any, value2 as any, optional precision as nullable number) as any
```

- **Value.Type**

返回给定值的类型。

```
Value.Type( value as any) as type
```

- **Variable.Value**

Variable.Value

Variable.Value(identifier as text) as any

● Web.Contents

将从 url 下载的内容返回为二进制。可以提供可选记录参数 options 以指定其他属性。记录可以包含以下字段:

Query : 以编程方式向 URL 添加查询参数而无需担心转义。

ApiKeyName : 如果目标站点对 API 密钥有一些概念，则此参数可用于指定必须在该 URL 中使用的密钥参数的名称(而不是值)。在凭据中提供实际密钥值。

Content : 如果指定此值，则会将 Web 请求从 GET 更改为 POST，并使用 Content 字段的值作为 POST 的内容。

Headers : 指定此值作为记录以向 HTTP 请求提供额外标头。

Timeout : 指定此值作为持续时间将更改 HTTP 请求的超时时间。默认值为 100 秒。

ExcludedFromCacheKey : 如果将此值指定为列表，将从缓存数据的计算中排除这些 HTTP 标头密钥。

IsRetry : 如果将此逻辑值指定为 true，则在提取数据时将忽略缓存中的任何现有响应。

ManualStatusHandling : 如果将此值指定为列表，将禁止其响应具有这些状态代码之一的 HTTP 请求进行任何内置处理。

RelativePath : 如果将此值指定文本，会在发出请求之前将其追加到基 URL。

Web.Contents(url as text, optional options as nullable record) as binary

● Web.Page

返回 HTML 文档的内容(分解为其组成结构)，以及完整文档的表示形式及其删除标记后的文本。

Web.Page(html as any) as table

● WebMethod.Delete

指定 HTTP 的 DELETE 方法。

- **WebMethod.Get**

指定 HTTP 的 GET 方法。

- **WebMethod.Head**

指定 HTTP 的 HEAD 方法。

- **WebMethod.Patch**

指定 HTTP 的 PATCH 方法。

- **WebMethod.Post**

指定 HTTP 的 POST 方法。

- **WebMethod.Put**

指定 HTTP 的 PUT 方法。

- **WebMethod.Type**

指定 HTTP 方法。

- **Xml.Document**

返回 XML 文档的内容作为层次结构表。

Xml.Document(contents as any, optional encoding as nullable number) as table

- **Xml.Tables**

返回 XML 文档的内容作为平展表的嵌套集合。

Xml.Tables(contents as any, optional options as nullable record, optional encoding as nullable number) as table