第 3 章　DAX 简介

现在你已了解了 PowerPivot for Microsoft Excel 2013 中的一些功能，是时候来学习些 DAX 语言基础了。PowerPivot 有用于定义计算表达式的自有语法。从概念上讲这和 Excel 表达式很相似，但其有特有的功能允许你对存储于多个表格中的数据创建更高阶的计算（PowerPivot 的语言被称为“数据分析表达式”语言，本书均使用缩略词 DAX）。

在本章中，你将学习 DAX 基础知识，以及如何用 DAX 解决一些业务场景中的典型问题。

3.1　理解 DAX 计算

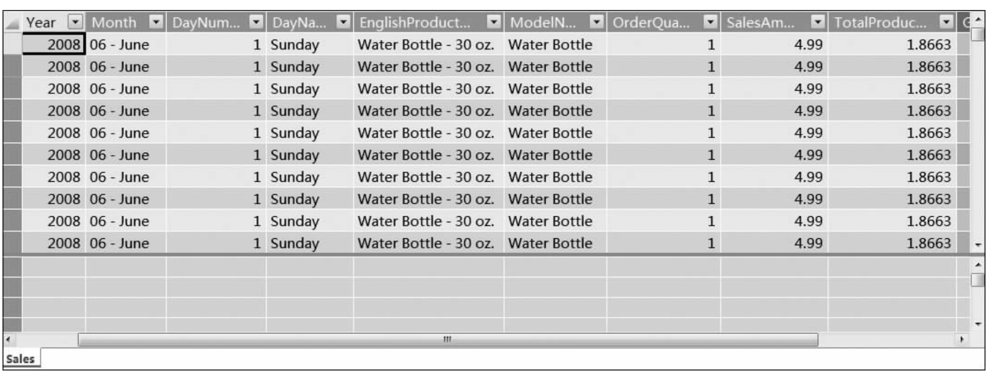
如同 Excel 一样，DAX 中的任何计算都从赋值运算符开始。主要区别在于 DAX 从不使用类似于 A1、C2 等等的单元格坐标。在 DAX 中总使用列名和表名来指定坐标。此外 DAX 并不支持 Excel 中所支持的“范围”概念：要有效地使用 DAX，你需要学习操作“列”和“表”。

注意：在 DAX 表达式中，无论是对于单行的一列，还是整个表格的一列，都可以得到一列值，也就是说无法访问表格内的某一行。要获得一个范围，则需要使用 DAX 函数对表格进行筛选，从而返回原始表格中对应于所需范围的某些行的子集。

要表达复杂公式，你需要学习 DAX 基础知识，其中包括语法、DAX 可处理的各种数据类型和基本运算符，以及如何引用列和表。接下来的几节将介绍这些概念。

3.1.1　DAX 语法

要理解 DAX 语法是如何工作的，一个相对简单的方式是从范例开始。假设有一个像图 3-1 所示的 PowerPivot 表，在该例中 SalesAmount 列和 TotalProductCost 列是有用的。你可以在配套的工作簿 CH03-01 Sales Example. xlsx 中找到这个数据模型。

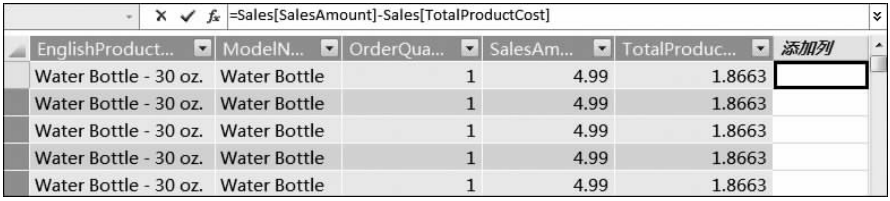


Year	Month	DayNum...	DayNa...	EnglishProduct...	ModelN...	OrderQua...	SalesAm...	TotalProduc...
2008	06 - June	1	Sunday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
2008	06 - June	1	Sunday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
2008	06 - June	1	Sunday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
2008	06 - June	1	Sunday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
2008	06 - June	1	Sunday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
2008	06 - June	1	Sunday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
2008	06 - June	1	Sunday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
2008	06 - June	1	Sunday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
2008	06 - June	1	Sunday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663
2008	06 - June	1	Sunday	Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663

Sales

图 3-1　我们将使用该表来展示 DAX 语法

假设要通过从 SalesAmount 减去 TotalProductCost 来计算毛利。需要在一个新列中编写如图 3-2 所示的 DAX 公式，你可以称之为 GrossMargin。



=Sales[SalesAmount]-Sales[TotalProductCost]

EnglishProduct...	ModelN...	OrderQua...	SalesAm...	TotalProduc...	添加列
Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663	
Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663	
Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663	
Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663	
Water Bottle - 30 oz.	Water Bottle	1	4.99	1.8663	

图 3-2　在公式栏中可以输入 GrossMargin 的定义

对于表格中的每一行，这个新公式都会自动重复，最终形成表格中的一个新列。在本例中，使用 DAX 表达式来定义一个计算列（在本章稍后，我们也将展示 DAX 用于定义计算字段）。DAX 表达式处理数值，并且还返回一个数值。

3.1.2　DAX 数据类型

在前述公式中你可看到如何处理数值。DAX 可以对以下 7 类数据类型开展计算：

- 整型；
- 实数；
- 货币型；
- 日期（日期时间）；
- TRUE/FALSE（布尔）；
- 字符串；
- BLOB（代表“二进制大对象”）。

PowerPivot 有个强大的数据类型处理系统，所以不必过多地担心数据类型：在写 DAX 表达式时，公式结果的类型取决于表达式中所用术语的类型。你必须要意识到这一点，如果从 DAX 表达式中返回的数据类型不同于预期类型，那么就必须检查在表达式本身中所用术语的类型。例如，如果一个求和项的术语是一个日期类型，其结果也会是一个日期，如果同一运算符用于整型，其结果也是一个整型。这就是所谓的运算符重载，在图 3-3 你可以看到这个特性的例子，其中 DatePlusOneWeek 列是通过对 Date 列中的值加上 7 计算得到的，其结果是一个日期。

[DatePlusOneWeek]　=Sales[Date] + 7

Date	DatePlusOneWeek	Year	Month	DayNum...	DayNa...
2008/6/1	2008/6/8	2008	06 - June	1	Sunday
2008/6/1	2008/6/8	2008	06 - June	1	Sunday

图 3-3　日期加上整型的结果是增加相应天数后的日期

除了运算符重载，PowerPivot 还能自动将字符串转换为运算符所需的数字，也能将数字转换成运算符所需的字符串。例如，如果使用 & 运算符来连接字符串，PowerPivot 自动将其参数转换成字符串。下面的公式

```
=5 & 4
```

返回字符串 54。而下面的公式

```
="5"+"4"
```

返回一个结果值为整型的数字 9。

如上所述，结果值取决于运算符而非源列，这是按照运算符的要求所做的转换。但是即便这种特性颇为方便，在 3.4 节中，你会看到在这些自动转换过程中可能会发生哪些类型的错误。

DAX 数据类型内部

DAX 数据类型可能为 Excel 用户所熟悉。然而，在 PowerPivot 数据模型中，有两类数据类型经常使用，我们需要看一看这两类的一些特殊情况。

“日期”数据类型

PowerPivot 在 datetime 数据类型中储存日期。这种格式内部使用浮点数，其中整数对应天数（以 1899 年 12 月 30 日为起点），而小数表示不足一天的部分（将小时、分、秒转换为小于 1 的天数）。因此，表达式为

```
=NOW()+1
```

对一个日期增加一天（正好是 24 小时），返回第二天的日期，以及同表达式自身所执行时间相同的小时、分钟、秒。

TRUE/FALSE 数据类型

TRUE/FALSE 数据类型是用于表达逻辑条件的。例如，以下表达式定义的计算列即为 TRUE/FALSE 类型：

```
=Sales[TotalProductCost]>Sales[Amount]
```

该数据类型的性质与 Excel 中的数据类型的性质相似，通常称之为布尔型数据类型。一般这种类型的一列对于用户而言是不可见的，仅内在地用于 DAX 计算。

3.1.3 DAX 运算符

DAX 中可用的运算符如表 3-1 所示。

表 3-1 运算符

运算符类型	符号	用　法	例　子
圆括号	()	参数的优先顺序和分组	(5＋2)＊3
算术符	＋ － ＊ /	加法 减法 乘法 除法	4＋2 5－3 4＊2 4/2

续表

运算符类型	符号	用　法	例　子
比较符	=	等于	[Country]="USA"
	<>	不等于	[Country]<>"USA"
	>	大于	[Quantity]>0
	>=	大于等于	[Quantity]>=100
	<	小于	[Quantity]<0
	<=	小于等于	[Quantity]<=100
文本连接	&	连接字符串	"Value is"&[Amount]
逻辑符	&&	且	[Country]="USA"&&[Quantity]>0
	||	或	[Country]="USA"||[Quantity]>0
	!	非	!([Country]="USA")

此外 DAX 函数还可使用逻辑运算符，其语法类似于 Excel 语法。例如：

```
AND ([Country]="USA", [Quantity]>0)
OR ([Country]="USA", [Quantity]>0)
NOT ([Country]="USA")
```

分别对应于

```
[Country]="USA" && [Quantity]>0
[Country]="USA"||[Quantity]>0
!([Country]="USA")
```

3.1.4　DAX 值

如你所见，可以直接在公式中使用值（例如，美国或 0）。当这些值以该方式使用时，称为文本，虽然文本使用简便，应考虑引用列的语法。基本的语法为

```
'table name' [列名]
```

表名可由单引号括起来。然而如果该名称不包含任何特殊字符（如空格），多数情况下可省略引号。例如，以下公式中可以省略引号：

```
tablename[列名]
```

另一方面，列名必须要括在方括号内。需注意的是，表名是可选的。如果省略表名，就仅搜索当前表格中的列名，也就是说计算列或度量所属的表格。然而我们强烈建议你始终指定完整名称（表和列），以避免任何混淆。

智能感知

当在 Excel 或 PowerPivot 中编写公式时，一个称为 IntelliSense 的特别帮助功能显示了公式中可使用的所有可能的函数名称和引用。当在 PowerPivot 窗口中编写公式时，如果输入左方括号，IntelliSense 仅显示当前表格中的列（你正在其中定义计算列或计算字段的表格）。如果输入一个表名中的首字母，IntelliSense 中显示表名和列名。

如图 3-4 所示，在向计算列公式中键入左方括号时，会显示出当前表格的列清单。

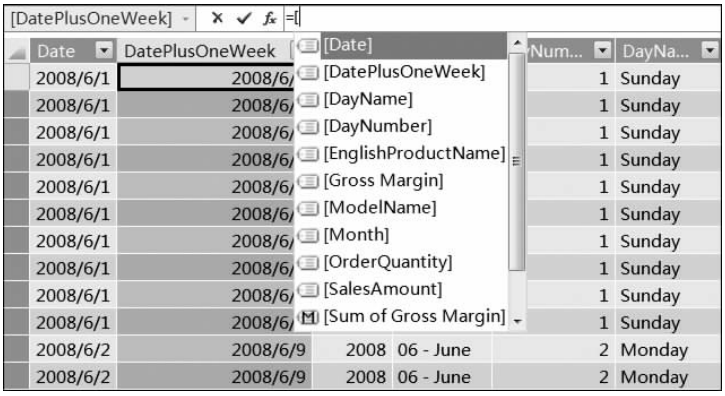


图 3-4　当向 PowerPivot 窗口输入左方括号时，IntelliSense 显示当前表格中的所有字段

3.2　了解计算列和计算字段

目前你已知道了 DAX 语法基础知识，还需要学习 DAX 中最重要的概念之一：计算列和计算字段之间的差异。由于你可用两种方式进行计算，这两种方式最初可能看起来很类似，但实际上差异非常大。了解它们之间的差异是发挥 DAX 能力的关键。

3.2.1　计算列

在第 2 章中，你学习了如何定义计算列。为此可使用功能区“设计”选项卡上的“添加”按钮，或者可简单地移动到最后一列（名为“添加列”）并开始编写公式。DAX 表达式已经插入到公式栏中，并且 IntelliSense 帮你编写该表达式。

计算列，就如同 PowerPivot 表格中任何其他列一样。可在数据透视表中的行区域、列区域、筛选器区域或值区域中使用。计算列所定义的 DAX 表达式在所属表格的当前行的环境下运行。对某列的任何引用都返回当前行中的该列的值。无法直接访问其他行的值。

注意：正如 3.6.1 节所看到的，DAX 函数可对整个表格聚合某列的值。获得行的子集的值的唯一方式，是使用 DAX 函数来返回一个表格，然后对其进行操作。通过该方式，可以对一个行范围来聚合列值，或者从另一行得到单个值并对其进行操作。使用 DAX，可引用一个表格并应用筛选器，以便检索结果只包括所需要的行。

计算列可轻松创建和使用。图 3-2 显示了如何定义毛利润列以计算毛利润总额：

```
[Gross Margin]=Sales[SalesAmount]-Sales[TotalProductCost]
```

计算列的表达式为每一行都计值，并将其结果存储于表格之中，就好像是从数据库中检索出来的列一样。计算列是 Excel 用户所熟悉的，因为他们也以非常类似的方式来处理 Excel 表中的列。

3.3 计算字段

你可能还记得第 2 章中当定义毛利润为一个值时，使用计算列处理很恰当；但如果要将毛利作为百分比进行计算，就需要定义一个计算字段。现在正是你对计算字段及所需语法加深理解的时候了。

计算字段是一个 DAX 表达式，使用的语法同计算列毫无二致；不同的是计值上下文。计算字段是在数据透视表中单元格上下文中进行计值，而计算列是在 PowerPivot 表格的行的层面进行计算。单元格上下文取决于用户在数据透视表中的选择。使用计算字段中的 SUM(SalesAmount)时，你的意思是对“数据透视表单元格底层的所有行”汇总；而在计算列中使用 Sales[SalesAmount]时，你的意思是 SalesAmount 在该行中的值。

当创建计算字段时你可以定义一个值，该值随着用户对数据透视表所应用的筛选器而变化。这样就可解决毛利率百分比计算的问题了。

要定义一个计算字段，可在选定数据透视表中的某个单元格时，通过单击 Excel 功能区 POWERPIVOT 选项卡上的“计算字段”按钮来选择“新建计算字段”的下拉菜单(见图 3-5)。

图 3-5　POWERPIVOT 功能区的“计算字段”选项

目前“计算字段”对话框(见图 3-6)已打开，可选择新计算字段所在的“表名”、“计算字段名称”和“说明”，最后是 DAX 公式和格式类别。

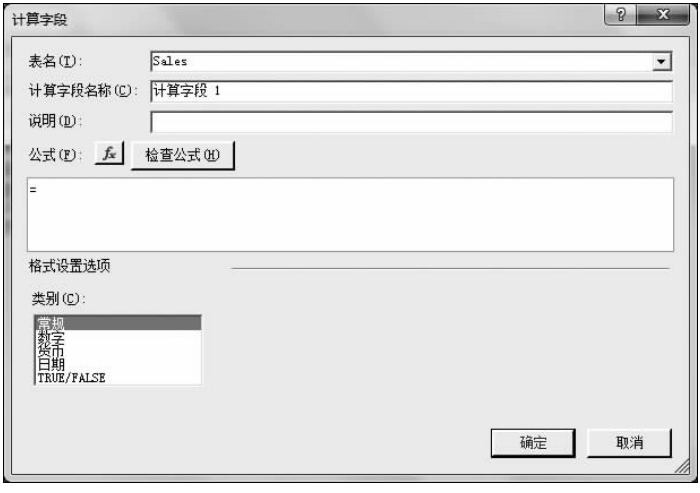


图 3-6　这里显示了计算字段窗口，用于创建新的计算字段

现在让我们着重了解定义新度量的公式。在初次尝试定义度量时，可能会使用与计算列中所用的相同 DAX 表达式，但出现如图 3-7 所示的错误。

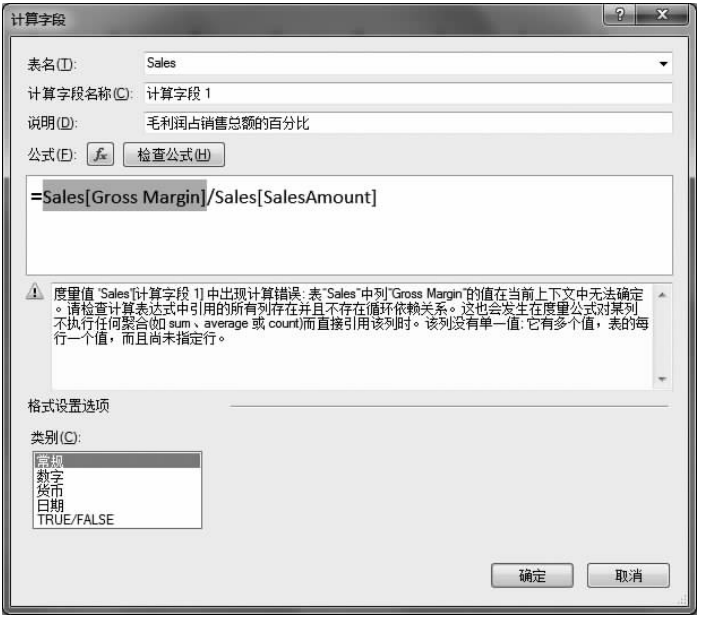


图 3-7　如果使用与计算列相同的 DAX 表达来定义 GrossMarginPerc(毛利润占销售总额的百分比)计算字段，你会得到一条错误信息计算字段

通过单击“检查公式”来显示错误消息。此错误的原因在于，该运算的上下文并非单个行，而是一组行。选定的区域已经被单元格所隐式定义，且需要在透视表中开展运算。在包含多个行的环境下，是无法引用列 Sales[SalesAmount]的值的，因为只有在单个行环境下使用该列时，该列仅有一个值。

要避免此类错误，你需要定义一个表达式，用毛利润总额除以销售额。该表达式使用 SUM 函数，对数据透视表当前选择的所有行进行聚合(正如之前所述，这里选择的是年份)。以下是计算字段的正确 DAX 表达式，如图 3-8 所示。

```
=SUM(Sales[Gross Margin])/SUM(Sales[SalesAmount])
```

计算列和计算字段之间的差异

看起来非常相似的计算列和计算字段却有着天壤之别。计算列的值是在数据刷新时进行计算的，并且使用了当前行作为上下文，计算列并不依赖于用户对数据透视表的操作。计算字段是对在当前单元格上下文中所定义的数据进行的聚合：根据单元格坐标相应地对源表进行筛选，并使用这些筛选器聚合并计算数据。换句话说，计算字段始终是对处于计值上下文中的数据进行聚合，基于此原因，其默认执行模式并不引用任何单一行。在第 7 章中将对计值上下文进一步解释。

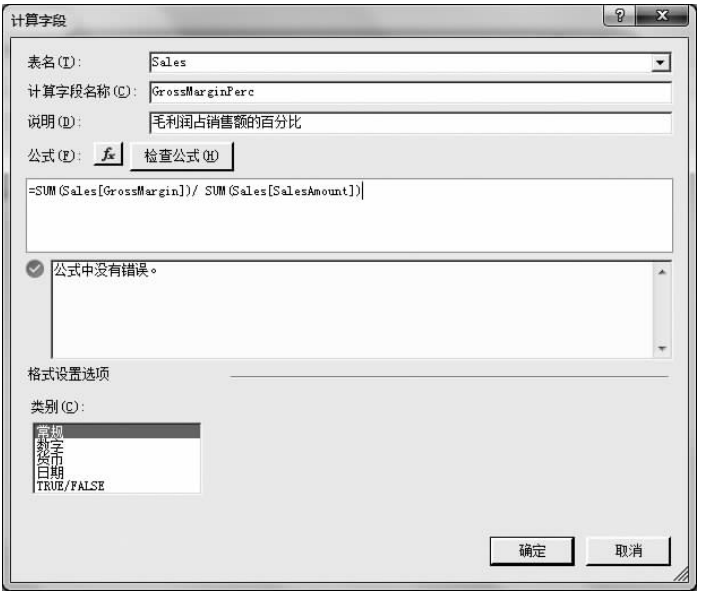


图 3-8　GrossMarginPerc 度量计算字段的正确定义

3.3.1　计算列和度量之间的选择

目前你已看到计算列和计算字段之间的差别，你可能想知道，应该何时使用计算列，何时使用计算字段？有时可以任选其一，但在大多数情况下要按所需的计算进行选择。要进行以下操作时，必须（在 PowerPivot 表格网格窗口）定义计算列：

- 将计算结果置于 Excel 切片器或者数据透视表的行或列中（而不是值区域）以查看计算结果。
- 定义一个同当前行严格绑定的表达式（例如，“价格 * 数量”无法对这两列的平均值进行计算）。
- 对文本或数字进行归类（例如顾客年龄度量值的范围，如 0～18，18～25，等等）。

但是要显示所得计算值时必须定义一个计算字段，因为这些计算值反映了用户的数据透视表选择，并可在数据透视表的值区域看到。例如：

- 当计算数据透视表所选的利润百分比时。
- 当计算按年份或区域划分的某个产品相对于所有产品的比率时。

某些计算，使用计算列和计算字段均能实现，即便在这些情况下也必须使用不同的 DAX 表达式。例如，可以用计算列定义 GrossMargin，如下：

```
=Sales[SalesAmount]-Sales[TotalProductCost]
```

但也可以用计算字段定义：

```
=SUM(Sales[SalesAmount])-SUM(Sales[TotalProductCost])
```

最终结果是完全一样的。在本例中建议使用计算字段，在查询时进行计值，这样做不消

耗内存和磁盘空间。然而，该因素只有在处理大数据集时才非常重要。当工作簿大小不成问题时，大可选用更舒适的方法。

> **交叉引用**
>
> 很明显，计算字段可以引用一个或多个计算列。反之虽不直观，倒也亦然：一个计算列可引用一个或多个计算字段。此时，它迫使在当前行所定义的上下文中对某个字段进行计算。此操作将计算字段的结果变换并合并成一列且不受用户操作的影响。显然仅某些操作可产生有意义的结果，因为通常计算字段高度依赖于用户在数据透视表中所作的选择。

3.4 处理 DAX 表达式中的错误

现在你已看到一些基本公式，在发生无效计算时你应学会如何适当地处理。由于所引用的数据对公式而言是无效的，DAX 表达式可能包含无效计算。例如，你可能遇到分母为 0 的情况，或遇到将非数值用于算术（如乘法）运算的情况。你必须学习默认情况下这些错误是如何得到处理的，以及如果想以特殊方式处理这些错误，如何阻止这些默认条件。

在学会如何处理错误之前，很值得对 DAX 公式计值时可能出现的各类错误进行描述。这些错误类型分别是

- 转换错误；
- 算术运算；
- 空值或缺失值。

3.4.1 转换错误

我们要分析的第一种错误类型是转换错误。正如 3.1.2 节所述，当运算符需要时，DAX 值可在字符串和数值之间自动转换。例如，以下 DAX 表达式均有效：

```
"10"+32=42
"10" & 32="1032"
10 & 32="1032"
DATE (2010,3,25)=3/25/2010
DATE (2010,3,25)+14=4/8/2010
DATE (2010,3,25) & 14="3/25/201014"
```

因为所运算的都是常量值，这些公式都正确。但以下公式又如何呢？

```
SalesOrders[VatCode]+100
```

因为此求和中的第一个运算符，是通过“列”获得的（本例中是个文本列），你必须确保该列中的所有值都是数字，以确保无论它们是否将被转换，表达式均能正确地计值。如果某些内容无法转换以满足运算符的要求，就会发生转换错误。以下是典型情况：

```
"1+1"+0=无法将字符串类型的值"1+1"转换为实数类型
DATEVALUE("25/14/2010")=类型不匹配
```

要避免这些错误，需要编写更复杂的 DAX 表达式来包含错误检测逻辑，以便截获错误并返回有意义的结果。

3.4.2 算术运算错误

第二种错误类型是算术运算错误，如分母为 0 或负数的平方根。这些类型的错误与转换无关，而是使用无效值来调用某个函数或使用运算符时所产生的算术运算错误。

在 PowerPivot 中，分母为 0 的情况需要特殊处理，因为这是种很不直观的方式（数学家除外）。当你对某个数值除以 0 时，PowerPivot 通常返回特殊的无穷大值。此外，在 0 除以 0 或无穷大除以无穷大的特例中，PowerPivot 返回特殊值 NaN（非数字）。这对 Excel 用户而言很奇怪，我们在表 3-2 中进行了总结。

表 3-2 分母为 0 时得到的特殊结果值

表达式	结果	表达式	结果
10/0	Infinity	0/0	NaN
7/0	Infinity	(10/0)/(7/0)	NaN

值得注意的是：无穷大和 NaN 不是错误，而是 PowerPivot 中的特殊值。事实上，如果你用一个数值除以无穷大，表达不会生成错误而是返回 0（注意，在下面的表达式中，7/0 的结果为无穷大）：

```
9954/(7/0)=0
```

除了这种特殊情况，当用某个错误参数（如负数的平方根）调用 DAX 函数时，也可能返回算术错误：

```
SQRT(-1)=函数开方的参数中包含错误的数据类型或者结果过大或过小
```

如果 PowerPivot 检测到类似错误，便会阻止 DAX 表达式的所有进一步计算，并提示错误。可以使用特殊 DAX 函数 ISERROR 来检查是否表达式会导致错误，你将在 3.5.3 节学习使用 ISERROR 函数。最后，在 PowerPivot 窗口中 NaN 这种特殊值可以显示，但是当显示在 Excel 数据透视表中时将显示为错误，并将被错误检测功能检测为错误。

3.4.2.1 空值或缺失值

第三类并非特殊错误条件，而是存在空值，当在计算中将空值与其他元素相结合时，可能会导致意想不到的结果或计算错误。你需要了解如何在 PowerPivot 中处理这些特殊值。

DAX 以同样方式处理缺失值、空白和空单元格：BLANK（空值）。空值没有真正的值，而是一种识别这些情况的特殊方式。在 DAX 表达式中通过调用 BLAND 函数可以得到空值，这不同于空字符串。例如，下面的表达式总是返回空值，这时在 PowerPivot 窗口显示为一个空单元格：

```
=blank()
```

就其本身而言，这种表达是没用的，但要返回一个空值时，BLANK 函数变得非常有用。例如，你可能要显示一个空单元格而非 0 值，下面的表达式计算销售交易的总折扣，如果折扣为 0 时将得到空单元格：

```
=IF (Sales[DiscountPerc]=0, BLANK(), Sales[DiscountPerc] * Sales[Amount])
```

如果 DAX 表达式包含一个空值，是不被视为错误的。此时，根据所需的计算可能会返回一个值，也可能返回空白。例如，下面的表达式

```
=10 * Sales[Amount]
```

只要销售金额是空值就会返回空值。换言之，只要乘法运算项中有一个或两个项是空值，运算的乘积就是空值。在 DAX 表达式中有几个其他算术和逻辑运算都会发生空值传递，如下面的例子所示：

```
BLANK()+BLANK()=BLANK()
10 * BLANK()=BLANK()
BLANK()/3=BLANK()
BLANK()/BLANK()=BLANK()
BLANK()||BLANK()=FALSE
BLANK() && BLANK()=FALSE
```

然而并非所有公式都会对空值进行传递。有些运算不传递空值，但返回的值取决于公式的其他项。这方面的情形包括加上空值，减去空值，除以空值，以及空值同一个有效值之间的逻辑运算。在下面的表达式中，你可以看到这些情况以及相应的结果：

```
BLANK ()-10=-10
18+BLANK ()=18
4/BLANK ()=Infinity
0/BLANK ()=NaN
FALSE||BLANK=FALSE
FALSE && BLANK=FALSE
TRUE||BLANK =TRUE
TRUE && BLANK=FALSE
```

Excel 中的空值

Excel 有多种方式处理空值。在 Excel 中，当求和或做乘法时，所有空值被视为 0；但如果空值是除法或逻辑表达式的一部分时，返回错误。

了解 DAX 表达式对空值的处理方式，在计算中使用 BLANK 返回一个空的单元格，这些都是控制 DAX 表达式结果的很重要的技能。正如在 3.4.3 节我们将学习到的，发现错误值或其他错误时，可以使用空值作为结果。

3.4.3 截获错误

既然已看到各式各样可能发生的错误，就应该学习一些技术来截获并纠正这些错误，至少显示一条包含有意义信息的错误消息。DAX 表达式中错误的出现往往取决于表达式本身引用的表和列中包含的值。所以你可能要控制这些错误条件的存在并返回一条错误消息。标准技术是去检查表达式是否返回一个错误，如果返回错误，用消息或一个默认值来替换错误。有一些 DAX 函数就是设计用来做这类工作的。

首先是 IFERROR 函数，与 IF 函数非常类似，然而并非计值 TRUE/FALSE 条件，而是检查表达式是否返回一个错误。这里你可以看到 IFERROR 函数的两个典型用途：

```
=IFERROR (Sales[Quantity] * Sales[Price], BLANK ())
=IFERROR (SQRT (Test[Omega]), BLANK ())
```

在第一个表达式中，如果 Sales[Quantity]（销售数量）或 Sales[Price]（销售价格）是不能转换成数值的字符串，那么返回的表达式为一个空单元格，否则返回 Quantity（数量）和 Price（价格）的乘积。

在第二个表达式中，只要 Test[Omega]列包含负数，结果就返回一个空单元格。

当这样使用 IFERROR 时，你遵循更一般的模式，需要使用 ISERROR 和 IF：

```
=IF (
    ISERROR (Sales[Quantity] * Sales[Price]),
    BLANK (),
    Sales[Quantity] * Sales[Price]
)
=IF (
    ISERROR (SQRT (Test[Omega])),
    BLANK (),
    SQRT (Test[Omega])
)
```

当必须返回的表达式与被检测错误的表达式是同一表达式时，应该使用 IFERROR 函数；你不必在两个位置重复表达式，这样得到的公式可读性更强且更易于后续修复且避免了引入错误。但是如果出现了错误，而又想返回不同的结果时，应该使用 IF 函数。

例如，ISNUMBER 可用于检测是否一个字符串（第一行中的价格）可以转换成数值，然后计算总额，否则，可以返回一个空单元格。

```
=IF (ISNUMBER (Sales[Price]), Sales[Quantity] * Sales[Price], BLANK ())
=IF (Test[Omega]>=0, SQRT (Test[Omega]), BLANK ())
```

第二个例子简单地检测 SQRT 参数有效与否，仅对正数求平方根，而对负数返回空值。

一个特定情况是对空值测试。ISBLANK 函数检测空值条件，如果参数为空值，返回 TRUE。当空值有意义而并非简单的 0 值时，这尤为重要。在下例中，如果销售交易中未指定重量，就使用产品默认的运送成本来计算销售交易的运送成本：

```
=IF (
    ISBLANK (Sales[Weight]),
    RELATED (Product[DefaultShippingCost]),
    Sales[Weight] * Sales[ShippingPrice]
)
```

如果仅简单地将产品重量和运费价格相乘，那么对于缺失重量数据的所有销售交易，将得到空的成本。

3.5 设置 DAX 代码格式

在继续解释 DAX 语言之前，DAX 函数还有一个非常值得学习的重要方面：格式化代码。DAX 是一种函数语言，也就是说无论多么复杂，DAX 表达式也只是一个调用一些参数的函数。代码复杂性反映为表达式的复杂性，这时表达式作为参数传递到函数的更外层。

出于这方面原因，看到 10 行甚至 10 行以上的表达式也很正常。一个 20 行的 DAX 表达式可能看起来挺奇怪但也正常，你会慢慢习惯的。随着公式长度与复杂性的增长，学习如何正确书写公式以便易于阅读变得至关重要。

还没有“官方”标准来规范 DAX 代码，但说明我们使用代码的标准很重要。我们的标准可能不尽完美，而且萝卜白菜各有所爱。在对格式化代码时，唯一需要记住的是：“不要把所有东西都放在同一行之中，如果在没意识到，你就陷入麻烦了！”

为说明格式缘何如此重要，这里展示一个将在第 12 章中使用的公式，该公式有点复杂，但还不是要创建的最复杂公式。如果未能以某种方式格式化，表达式看起来就如下所示：

```
IF (COUNTX (BalanceDate, CALCULATE (COUNT(Balances[Balance]),
ALLEXCEPT(Balances, BalanceDate[Date])))>0, SUMX (ALL (Balances[Account]),
CALCULATE(SUM(Balances[Balance]), LASTNONBLANK (DATESBETWEEN
(BalanceDate[Date], BLANK(),LASTDATE(BalanceDate[Date])), CALCULATE
(COUNT(Balances[Balance])))))), BLANK ())
```

试图了解这个公式计算了什么，几乎是不可能的，因为不知道最外层函数是什么，不同的函数调用是如何进行嵌套以创建完整执行流程的。我们已经看到太多由客户以这种方式编写公式的例子，在某些时候，为了有助于理解为何公式返回错误的结果，应该做的第一件事是规范化表达，只有这样我们才能开始工作。

同一个表达式在适当设置格式后，看起来如下：

```
=IF (
COUNTX (
       BalanceDate,
       CALCULATE (
           COUNT (Balances[Balance]),
           ALLEXCEPT (Balances, BalanceDate[Date])
      )
  )>0,
```

```
    SUMX (
        ALL (Balances[Account]),
        CALCULATE (
            SUM (Balances[Balance]),
            LASTNONBLANK (
                DATESBETWEEN (
                    BalanceDate[Date],
                    BLANK (),
                    LASTDATE(BalanceDate[Date])
                ),
                CALCULATE (COUNT (Balances[Balance]))
            )
        )
    ),
    BLANK ()
)
```

同样的代码，但现在更容易识别出 IF 的 3 个参数，而且最重要的是，遵循着由缩进行自然凸显的组块，看看是这些组块是如何建立其完整流程的。是的，代码是很难读，但现在仅是 DAX 的问题，而不再是不规范的问题，你不久将学会如何阅读和管理 DAX。

因此，这里有一套我们在本书和相关工作簿中使用的规则：

- 关键词（如 IF、COUNTX 和 CALCULATE）总是与任何其他项之间用空格分离，而且它们总是以大写字母书写。
- 所有列引用，都是以 TableName[ColumnName]方式书写，表名和左方括号之间没有空格。
- 逗号始终在一个空格之后，绝非在一个空格之前。
- 如果公式仅一行，那么无须应用其他规则。
- 如果公式不止一行，那么以下情况适用：
  - 函数名称本身占据一行，后跟左括号。
  - 所有参数均为独立的行，缩进 4 个空格并在表达式后附带一个逗号。
  - 右括号与函数调用对齐，并且本身占据一行。

这是我们使用的基本规则。如果你能找到一种最适于自己阅读表达式习惯的方式，尽可使用。规范化的目标是使公式更易于阅读，所以应该使用最适合你的方式。当定义你自己的公式规则时，要牢记最重要的是你能尽快地查看到错误。否则像前面所示的没有设置格式的代码中，如果 DAX 提醒你缺少一个右括号，你将很难发现错误究竟在哪儿。另一方面，规范的代码能使你更易于了解括号与函数调用是如何匹配的。

规范化 DAX 的几点帮助

格式化 DAX 并不容易，因为需要在一个文本框中用小字体来书写，不幸的是 Excel 并未提供编辑器。不过以下提示对编写 DAX 代码有帮助：

- 如果想使文字变大，可以使用 Ctrl＋鼠标滚轮增加字体大小来使代码更容易看清。

- 如果想对公式添加新行，可以按 Shift＋Enter 键。
- 如果在文本框中编辑真的很难，可以将代码复制到另一编辑器（如记事本）中进行更改，然后再次将公式粘贴到文本框中。

最后，在查看 DAX 表达式时，乍一看很难理解究竟是计算列还是计算字段。因此，本书使用等号（＝）来定义计算列，使用赋值运算符（:＝）来定义计算字段：

```
CalcCol=SUM (Sales[SalesAmount])          是一个计算列
CalcFld :=SUM (Sales[SalesAmount])        是一个计算字段
```

3.6 常用的 DAX 函数

目前你已经了解了 DAX 的基本原理和对错误情况如何处理，下面简要介绍最常用的函数和表达式。编写 DAX 表达式往往同 Excel 表达式类似，因为很多函数虽不尽相同，却也相似。多亏了以前的 Excel 知识，Excel 用户经常发现使用 PowerPivot 非常直观。在本章节会看到一些最常用的 DAX 函数，你很可能用它们来建立自己的 PowerPivot 数据模型。

在配套工作簿 CH03-02-Aggregation Functions. xlsx 中可以看到本节中的所有公式。

3.6.1 聚合函数

几乎每个 PowerPivot 数据模型都需要聚合数据。DAX 提供了一组函数，用于聚合表格中列的值并返回单个值，这组函数称为聚合函数。例如，下列表达式：

```
=SUM (Sales[Amount])
```

计算销售表的金额列中所有数值的总和。当用在计算列中时，该表达式聚合了销售表中的所有行；但当用在度量中时，该表达式仅考虑被数据透视表中的切片器、行、列和筛选器条件筛选后的那些行。

4 个聚合函数（SUM、AVERAGE、MIN 和 MAX）仅对数值操作。无论是名称上还是习性上，这些函数都与相应的 Excel 函数相同：运算中的任何非数值数据皆被忽略。在 PowerPivot 中，只有当这些作为参数传递的列为数值或日期类型时，这些函数才起作用。在图 3-9 中可以看到一个由聚合函数定义的计算字段的例子。

	列标签				
Values	2005	2006	2007	2008	总计
SUM of SalesAmount	3,266,373.66	6,530,343.53	9,791,060.30	9,770,899.74	29,358,677.22
AVG of SalesAmount	3,224.46	2,439.43	400.57	302.83	486.09
MIN of SalesAmount	699.10	699.10	2.29	2.29	2.29
MAX of SalesAmount	3,578.27	3,578.27	2,443.35	2,443.35	3,578.27

图 3-9　在此数据透视表中，可以看到聚合了 SalesAmount 值的使用统计函数的计算字段

如同 Excel 公式，DAX 对这些函数提供了可选的语法使得列计算中既可以包含数值，也可以包含非数值型的值，如文本列。这些语法简单地在函数名后增加了后缀 A，以得到与 Excel 中同样公式的相同名称和习性。然而，只有当列中包含 TRUE/FALSE 值时是有用

的，因为 TRUE 计值为 1，FALSE 为 0。文本列的任何值都视为 0。空单元格在计算中不予考虑。因此即使这些函数可在非数值列中使用而不返回错误，其结果也并不总与 Excel 的完全相同，因为 Excel 无法将文本列自动转换为数值。这些函数命名为 AVERAGEA、COUNTA、MINA 和 MAXA，图 3-10 显示了其在度量中用法的一个例子，该度量对同一工作表所示的示例表的 TRUE/FALSE 列进行运算。该表用作 PowerPivot 中的一个链接表，屏幕截图的下方是基于 PowerPivot 数据的数据透视表。

尽管这些聚合函数名称相同，但在 DAX 和 Excel 中使用时却有所区别。在 PowerPivot 中，列是有类型的，其类型决定了该列中聚合函数的行为。Excel 对每个单元格设置一个类型，而 PowerPivot 为每列设置一个类型。PowerPivot 中每列的类型均明确定义，并以表格式的形式（技术上称为关系型数据）处理数据，而 Excel 中缺乏对类型明确定义，Excel 公式是对异质单元格值的处理。如果 PowerPivot 中的一列是数值型，那么所有值就仅能为数值或空单元格。如果某列为文本类型，即使该文本可以转换为一个数值，这些 DAX 函数都视同该列始终为 0 来运算（COUNTA 除外）。而在 Excel 中，值是以逐个单元格为基础来视为数值的。出于此原因，这些 DAX 函数对于文本类型的列而言不起作用。

在以 A 为后缀的函数组中仅 COUNTA 有意义。其返回的单元格数值并非空值，且适于任何类型的列。如果对某个包含空值的列中的所有单元格的计数感兴趣，可以使用 COUNTBLANK 函数。最后，如果想对某列的所有单元格计数，而无论该列中内容如何，可以通过调用 COUNTROWS 函数来获得所要计算的表格中的行数（它以表格为参数，而非列）。换句话说，对同一表格的同一列，COUNTA 和 COUNTBLANK 之和总是等于这个表格中的行数，如图 3-11 所示。

Category	Produc	IsAct
Drink	Milk	TRUE
Drink	Soda	FALSE
Drink	Cofee	
Snacks	Chips	FALSE
Snacks	Peanuts	FALSE

	列标签		
Values	Drink	Snacks	总计
AVERAGEA Active	0.5	0	0.25
MINA Active	0	0	0
MAXA Active	1	0	1
COUNTA of IsActive	2	2	4

图 3-10　TRUE/FALSE 在以 A 为后缀的统计函数中计值为 1/0

Category	Produc	IsAct
Drink	Milk	TRUE
Drink	Soda	FALSE
Drink	Cofee	
Snacks	Chips	FALSE
Snacks	Peanuts	FALSE

	列标签		
Values	Drink	Snacks	总计
COUNTROWS of Table	3	2	5
COUNTBLANK of IsActive	1		1
COUNTA of IsActive	2	2	4

图 3-11　COUNTROWS 函数等同于同一列的 COUNTA 和 COUNTBLANK 之和

```
COUNTROWS(Sales)=COUNTA (Sales[SalesPersonID])
                    +COUNTBLANK (Sales[SalesPersonID])
```

所以可以使用下面 4 个函数来对一列或一个表格中的元素数量来计数：

- COUNT 只能对数值列操作。
- COUNTA 可对任何类型的列操作。
- COUNTBLANK 返回一列中的空单元格数。
- COUNTROWS 返回一个表格中的行数。

还有一个非常重要的计数函数——DISTINCTCOUNT，它对一列中非重复记录的条

目数量计数，即对此列中唯一参数计数。DISTINCTCOUNT 将空值作为可能值之一进行计数。因此如图 3-12 所示，类别 Drink 的 IsActive 列的 DISTINCTCOUNT 结果为 3，因为空值也是被计数的值。

Category	Produc	IsAct
Drink	Milk	TRUE
Drink	Soda	FALSE
Drink	Cofee	
Snacks	Chips	FALSE
Snacks	Peanuts	FALSE

	列标签		
Values	Drink	Snacks	总计
COUNTROWS of Table	3	2	5
COUNTBLANK of IsActive	1		1
COUNTA of IsActive	2	2	4
DISTINCTCOUNT isActive	3	1	3

图 3-12　DISTINCTCOUNT 将空值作为有效值来计数

注意：DISTINCTCOUNT 是在 2012 年版 PowerPivot 中引入的函数。要在以前版本的 PowerPivot 中计算某列的非重复记录的条目数，必须使用 COUNTROWS(DISTINCT(ColName))。两种模式返回同一结果，但 DISTINCTCOUNT 更易于阅读且仅需调用一个函数。

最后一组统计函数可对表格的每一行应用一个表达式，然后对该表达式进行一次聚合运算。尤其当要使用不同的相关表格中的列进行计算时，这组函数极为有用。例如，如果一个销售表格包含了所有的销售交易，且一个相关产品表格中包含了产品有关的信息，包括其成本，可以用以下表达式定义计算字段来计算销售交易总内部成本：

```
Cost:=SUMX (Sales, Sales[Quantity] * RELATED (Product[StandardCost]))
```

此函数对销售表格中的每一行都计算了数量(来自销售表格)和来自相关销售表格的标准成本的乘积，并返回所有这些计算值的汇总。

一般而言，所有以 X 结尾的聚合函数都以如下的方式运算：对表格(第一个参数)中的每一行计算一个表达式(第二个参数)，并返回一个结果，应用这些结果到对应的聚合函数(SUM、MIN、MAX 或 COUNT)从而得到最终的结果。

计值上下文对于理解这种计算如何工作非常重要。你将在第 7 章中对这种习性有更深刻的了解。可用的 X 后缀函数包括 AVERAGEX、SUMX、COUNTX、COUNTAX、MINX 和 MAXX。

3.6.2　逻辑函数

有时你可能想在表达式中建立一个逻辑条件，例如依据不同的列值实施不同计算或截获某个错误情况。在这些情况下可使用 DAX 中的某个逻辑函数。在 3.4 节中，你学到了这组函数中最重要的两个函数，即 IF 和 IFERROR。这些函数都非常简单，按其名字所表达的意思来运算：AND、FALSE、IF、IFERROR、SWITCH、NOT、TRUE 和 OR。例如，如果想在只有当价格列包含正确的数值型值时，才用价格乘以数量来计算金额，可以使用以下模式：

```
Amount:=IFERROR (Sales[Quantity] * Sales[Price], BLANK ())
```

如果未使用 IFERROR，且价格值包含无效数字，计算列的结果将出错，因为如果某一行生成一个计算错误，该错误将传递到整列。但是，IFERROR 的使用可截获错误并将其替换为空值。

逻辑函数中的 SWITCH 也很有意义，当某列含有的非重复值的条目数较小时(很多值

都是重复值),要依据这几个值来得到不同行为时尤为有用。例如,在 DimProduct 表格的尺寸包含 L、M、S 和 XL,你可能想对该值解码以让它变成更富有含义的列。可通过使用嵌套的 IF 调用来获得结果:

```
SizeDesc :=
    IF (DimProduct[Size]="S", "Small",
        IF (DimProduct[Size]="M", "Medium",
            IF (DimProduct[Size]="L", "Large",
              IF (DimProduct[Size]="XL", "Extra Large", "Other"))))
```

要表达相同公式,使用 SWITCH 是一种更方便的方式:

```
SizeDesc:=
    SWITCH (
        DimProduct[Size],
      "S", "Small",
      "M", "Medium",
      "L", "Large",
      "XL", "Extra Large",
      "Other"
  )
```

该表达式中的代码更易读,从内在讲 SWITCH 语句其实会翻译为嵌套的 IF 语句,所以计算速度不那么快。

提示:由于 SWITCH 可转换为一组嵌套的 IF 语句,第一个匹配的 IF 语句优先级最高,使用这种模式,可用同一个表达式测试多个条件:

```
SWITCH (
TRUE (),
    DimProduct[Size]="XL" && DimProduct[Color]="Red", "Red and XL",
    DimProduct[Size]="XL" && DimProduct[Color]="Blue", "Blue and XL",
    DimProduct[Size]="L" && DimProduct[Color]="Green", "Green and L"
)
```

使用 TRUE 作为第一个参数,意思是"当条件计值为真时,返回第一个结果"。

3.6.3 信息函数

每当需要分析表达式的类型时,可使用 DAX 中的某个信息函数。所有这些函数返回一个 TRUE/FALSE 值,并可用于任何逻辑表达式。这些信息函数是 ISERROR、ISBLANK、ISLOGICAL、ISNONTEXT、ISNUMBER 和 ISTEXT。

重要的是要注意,当表格的某列作为参数传递时,ISNUMBER、ISTEXT 和 ISNONTEXT 函数总是返回 TRUE 或 FALSE,这取决于该列的数据类型以及每个单元格的空条件。在图 3-13 中,可以看到 Price 列(此列为文本类型)是如何影响计算列的结果的。

```
ISBLANK        =ISBLANK (Sales[Price])
```

```
ISNUMBER       =ISNUMBER (Sales[Price])
ISTEXT         =ISTEXT (Sales[Price])
ISNONTEXT      =ISNONTEXT (Sales[Price])
ISERROR        =ISERROR (Sales[Price]+0)
```

Product	Price	ISBLANK	ISTEXT	ISNUMBER	ISNONTEXT	ISERROR
Bike	100	FALSE	TRUE	FALSE	FALSE	FALSE
Clock		TRUE	FALSE	FALSE	TRUE	FALSE
Hat	15	FALSE	TRUE	FALSE	FALSE	TRUE
Notebook	399	FALSE	TRUE	FALSE	FALSE	FALSE
Gadget	N/A	FALSE	TRUE	FALSE	FALSE	FALSE

图 3-13　信息函数的结果是基于列的类型的

你可能会想，ISNUMBER 是否可与一个文本列一同使用，以检查是否有可能转换为数值？不幸的是无法使用该方法；如果想测试某个文本值可否转换为数值，必须尝试转换并且要处理不能转换所产生的错误。例如，要测试是否 Price 列（该列为字符串类型）包含有效数字，你必须编写：

```
IsPriceCorrect=ISERROR (Sales[Price]+0)
```

以使得 ISERROR 函数得到 TURE 的结果为例，DAX 试图对价格加上 0 以强制将文本值转换为数值。对于 N/A 价格值转换失败，此时 ISERROR 结果为 TRUE。

假设你尝试使用 ISNUMBER 函数，表达式如下：

```
IsPriceCorrect=ISNUMBER (Sales[Price])
```

在这种情况下，由于元数据中的 Price 列并非数字而是字符串，因此只会得到 FALSE 结果。

3.6.4　数学函数

DAX 中的数学函数与 Excel 中的数学函数是非常类似的，语法和功能相同。最常用的数学函数有 ABS、EXP、FACT、LN、LOG、LOG10、MOD、PI、POWER、QUOTIENT、SIGN 和 SQRT。随机函数包括 RAND 和 RANDBETWEEN。最后还有四舍五入函数，这需要举一些例子，可使用几种方法来得到相同的结果。下面的几个函数的结果如图 3-14 所示。

```
FLOOR          =FLOOR (Tests[Value], 0.01)
TRUNC          =TRUNC (Tests[Value], 2)
ROUNDDOWN      =ROUNDDOWN (Tests[Value], 2)
MROUND         =MROUND (Tests[Value], 0.01)
ROUND          =ROUND (Tests[Value], 2)
CEILING        =CEILING (Tests[Value], 0.01)
ISO.CEILING    =ISO.CEILING (Tests[Value], 0.01)
ROUNDUP        =ROUNDUP (Tests[Value], 2)
INT            =INT (Tests[Value])
FIXED          =FIXED (Tests[Value], 2, TRUE)
```

从图中可以看出，FLOOR、TRUNC 和 ROUNDDOWN 非常相似，只是在指定的小数点位进行四舍五入的方式上有所区别。CEILING 和 ROUNDUP 在结果方面非常相似，但方式正相反。MROUND 和 ROUND 函数进行舍入的方式略有不同。最后重要的是要注意，FLOOR 和 MROUND 对不可用于负数，而其他函数可用于负数。

Test	Value	CEILIN	FLOOR	ROUND	TRUNC	ROUNDI	ROUNDU	INT	MROUNI	FIXED	ISO.CEILING
A	1.12345	1.13	1.12	1.12	1.12	1.12	1.13	1	1.12	1	1.13
B	1.265	1.27	1.26	1.27	1.26	1.26	1.27	1	1.26	1	1.27
C	1.265001	1.27	1.26	1.27	1.26	1.26	1.27	1	1.27	1	1.27
D	1.499999	1.5	1.49	1.5	1.49	1.49	1.5	1	1.5	1	1.5
E	1.51111	1.52	1.51	1.51	1.51	1.51	1.52	1	1.51	1	1.52
F	1.000001	1.01	1	1	1	1	1.01	1	1	1	1.01
G	1.999999	2	1.99	2	1.99	1.99	2	1	2	1	2

图 3-14　几种四舍五入函数总结

3.6.5　文本函数

DAX 中几乎所有可用的文本函数都与 Excel 中的非常相似，仅有少数例外。DAX 的文本函数有 CONCATENATE、EXACT、FIND、FIXED、FORMAT、LEFT、LEN、LOWER、MID、REPLACE、REPT、RIGHT、SEARCH、SUBSTITUTE、TRIM、UPPER 和 VALUE。这些函数可用于操控文本并从包含多个值的字符串中提取数据。例如在图 3-15 中，你可以看到从某个字符串中提取第一个和最后一个名字的例子，该字符串包含由逗号分隔的值，位于字符串中间的是我们想去除的称谓。

Name	Comma1	Comma2	FirstLastName	SimpleConversion
Russo, Mr., Marco	6	11	**Marco Russo**	Marco Russo
Ferrari, Mr., Alberto	8	13	**Alberto Ferrari**	Alberto Ferrari
Ferrari, Alberto	8		**Alberto Ferrari**	Ferrari, Alberto Ferrari

图 3-15　这里，你可以看到一个例子，使用文本函数来抽取姓和名

从通过计算两个逗号的位置开始，随后使用这些数字来提取文本中的右侧部分。SimpleConversion 列执行了一个公式，如果存在字符串中的逗号少于两个的情况（如果根本就没有逗号将提出错误）可能会返回错误值，而 FirstLastName 列执行了更复杂的表达式，使得在缺少逗号的情况下仍然成功。事实上在图 3-15 中可以看到，在 SimpleConversion 列中最后一行显示了不正确的值，而 FirstLastName 列显示了正确的转换。

```
Comma1=IFERROR (FIND (",", People[Name]), BLANK ())
Comma2=IFERROR (FIND (",", People[Name], People[Comma1]+1), BLANK ())
SimpleConversion=MID (People[Name], People[Comma2]+1, LEN (People[Name]))
                    & " " & LEFT (People[Name], People[Comma1]-1)
FirstLastName=TRIM (
                     MID (
                         People[Name],
                         IF (
                             ISNUMBER (People[Comma2]),
                             People[Comma2],
```