优客服数据报表查询语法

V3.6

（MDX 用户指南）

MDX 用户指南

1. 介绍

MDX 为 MultiDimensional Expressions 的缩写，多维表达式，是标准的 OLAP 查询语言。支持多维对象与数据的定义和操作。MDX 在很多方面与结构化查询语言 (SQL) 语法相似，但它不是 SQL 语言的扩展；事实上，MDX 所提供的一些功能也可由 SQL 提供，尽管不是那么有效或直观。

如同 SQL 查询一样，每个 MDX 查询都要求有数据请求（SELECT 子句）、起始点（FROM 子句）和筛选（WHERE 子句）。这些关键字以及其它关键字提供了各种工具，用来从多维数据集析取数据的特定部分。MDX 还提供了可靠的函数集，用来对所检索的数据进行操作，同时还具有用用户定义函数扩展 MDX 的能力。

MDX 为多维数据库提供了表达式查询语法，用于查询 Cube 数据，并提供了许多强大的分析函数，用于支持常用的 OLAP 分析。

R3 Query V3.0 OLAP 支持 MDX 语法，提供了大多数 MDX 函数。

2. 基本概念

2.1 Cube 结构

在介绍 MDX 之前 ,我们简单介绍一下 cube 结构,理解 cube 结构是学习 MDX 的前提。

Cube 是联机分析的关键。它们是一种多维结构，包括原始事实数据、聚合数据，这些数据聚合允许用户快速进行复杂查询。

Cube 包含两个基本的概念：维度和度量。

维度(Dimension): 维度提供了分类描述,表示一类分析角度,用户通过维度来分析度量数据.

度量(Measures): 度量表示用来聚合分析的数字信息,如数量,销售金额等.

重要: 度量的集合组成了一个特殊的维度,叫做” Measures”.

一个维度可以包含级别的层级结构,级别（ Level ）表示特定的分类.比如,地区维度可以包含级别层级:Country、State、City。每个级别比它的父级别在数据粒度上更加细粒度。

又比如：一个时间维可能包含级别：年、季、月。

成员，是最重要的概念之一。一个成员是维度（包括度量维 Measures）上的一个项目值，时间维度上“年”级别的成员可能有 2000、2001，月成员有 1、2、3 等等。

计算成员是一种运行时通过特殊表达式动态计算的成员。计算成员可以定义为度量。计算成员不影响现有的 cube 数据，它基于 cube 数据，通过各种数学表达式和各种函数定义，可以创建复杂的表达式。任何动态分析功能，都可以通过计算成员实现，比如实现占比、同期比等等。

2.2 示例 Cube 数据库

示例 OLAP 数据库结构

维　度		
维度名	级别	描述
Customers	Country State City Name	顾客的地理分布层级
Education Level	Edication Level	顾客教育水平，一个级别，扁平结构
Gender	Gender	顾客性别
Marital Status	Marital Status	顾客婚姻状况
Product	Product Family	产品维,6 个级别

	Product Department Product Category Product SubCategory Brand name Product Name	
Promotion Media	Media Type	促销媒体
Store	Store Country Store Sate Store City Stoe Name	商店的地理分布层级
Stoe Size in SQFT	Store Square Feet	商店面积
Store Type	Store Type	商店类型
Time	Years Quarters Months	时间维
Yearly Income	Yearly Income	顾客年收入

度量	
度量名	描述
Unit Sales	Number of units sold
Store Cost	Cost of goods sold
Store Sales	Value of sales transactions

Sales Count	Number of sales transactions
Store Sales Net	Value of sales transactions less cost of goods sold
Sales Average	Store sales/sales count

2.3 表达式表示法

维度、级别、成员等，一般用唯一名称 UniqueName 来标示，可以用[]包围 name，如果 name 有空格或者以数字开头，必须使用[],否则可以忽略。UniqueName 是根据层级结构表示的一种方法。即递归显示出祖先的名称。

维度(Dimension)：维度直接用[]包围。Product 的唯一名称为[Product]或 Product，维度 Education Level 的唯一名称为[Education Level]。度量维为[Measures]。

级别(Level)：级别的 UniqueName 为[维度名称].[级别名称]，如[Product].[Product Family]，同样，如果没有空格，[]可以省略，如 Product.[Product Family]。

成员(Member)：成员的 UniqueName 格式为[维度].(Parent Member UniqueName).[Member Name]，如上面时间维上的 2003 年 2 月份的 UniqueName 为[Time].[1].[2]，中间的 1 为 1 季度，因为该维度的结构为年、季、月。

度量(Measure):度量实际上是属于度量维的成员。如度量 Unit Sales 的 UniqueName 为[Measures].[Unit Sales]。

UniqueName 是 OLAP 元素内部的表示法，在 MDX 查询语言中，可以使用 UniqueName 来表示元素。同时，MDX 还提供模糊和其它等价的元素标示方式。表现在：

1) 省略维度名标示级别，如果一个维度的级别名称在整个 Cube 中是唯一的，那

么可以省略维度名来查询级别。如 Product.[Product Family]可以写成[Product Family]。

2) 省略维度名标示成员，如果省略维度名，可以标示该维度最高级别的成员，如[Time].[2000]可以写成[2000], [Measures].[Unit Sales]可以写成[Unit Sales].

3) 成员挂在级别下，即成员不一定要写成[维度].(Parent Member UniqueName).[Member Name]，可以写成[维度].[级别].[Member Name].如 2000 年 3 月可以写成[Time].[Months].[3]

模糊查询表示法,都基于不会重复的假设,如果有重复的元素,取第一个查找到的元素作为查询结果,可能发生错误.因此,建议使用完备的表示法.

2.4 元组、集合

元组用于定义来自多维数据集的数据切片 ;它由来自一个或多个维度的单个成员的有序集合组成。元组内不能包含来自同一个维度的多个成员.元组用()包围.如:

(时间.[下半年])

(时间.[下半年],[产品].[手机].[Nokia])

如果一个元组是由单个维度的成员组成,那么可以不用()包围,即(时间.[下半年])可表示成:时间.[下半年]

集合(set)是零个、一个或多个元组的有序集合。集合最常用于在 MDX 查询中定义轴维度和切片器维度，并且同样可能只具有单个元组或可能在某些情况下为空。在 MDX 语法中，元组用花括号括起来以构造集合，下面的示例显示具有两个元组的集合：

{ (时间.[上半年], 路线.非陆地.航空), (时间.[下半年], 路线.非陆地.海路) }

一个集合可包含同一个元组不止一次的出现。下面的集合是可接受的：

{ 时间.[下半年], 时间.[下半年] }

集合指以元组表示的一组成员组合，或指集合中的元组所代表的单元中的值，视集合使用的上下文而定。

注意，单个元组的集合不等于元组。如{时间.[下半年]}不等于 时间.[下半年].

在 MDX 语法中，很多函数语义中包含元组和集合，作为参数或者返回值。

3. MDX 查询与语义

3.1 MDX 基本查询

先看看 MDX 基本语法结构：

```
SELECT [axis specification] ON COLUMNS,[axis specification] ON ROWS FROM [cube name] WHERE [slicer specification]
```

基本的 MDX SELECT 语句包含一个 SELECT 子句和一个 FROM 子句，以及一个可选的 WHERE 子句。

[axis specificatioin]可以看成是轴的成员选择。[slicer specification]表示切片上的成

员，可以看成过滤信息，[slicer specification]可选，如果没有指定，取系统默认的维度成员作为切片。

该查询语句返回一个二维表格数据，可以显示为交叉表形式。

```
SELECT Measures.Members ON COLUMNS,[Store].Members ON ROWS FROM [Sales]
```

这里 Measures 表示度量维，[Store]表示 Store 维，[]可选，如果维度名称有空格，则需要用[]包围。Members 函数施加在维度上返回该维度的所有成员。该查询显示所有商店的度量值。二维表格如下：

	Store Sales	**Unit Sales**
深圳分店	200000.00	50000
上海分店	300000.00	60000

3.2 轴维度和切片器维度

当设计多维表达式（MDX）查询时，应用程序一般查看多维数据集并将维度集合划分为两个子集：

- 轴维度，为多个成员检索数据的维度。
- 切片器维度，为单个成员检索数据的维度。

因为轴维度和切片器维度都可从要查询的多维数据集的多个维度构造，所以用这些术语将要查询的多维数据集使用的维度与在由 MDX 查询返回的多维数据集中创建的维度区分开。

例如，假定存在名为 TestCube 的多维数据集，具有两个名为 Route 和 Time 的简单维度。因为多维数据集的度量值是 Measures 维度的一部分，所以该多维数据集总共有三个维度。查询要提供一个矩阵，可以在该矩阵内跨路线和时间比较 Packages 度量值。

在下面的 MDX 查询示例中，Route 和 Time 维度用作轴维度，Measures 维度用作切片器维度。Members 函数表明要用于构造集合的维度或级别的成员，而不必在 MDX 查询中显式声明给定维度或级别的各个成员。

```
SELECT
    { Route.nonground.Members } ON COLUMNS,
    { Time.[1st half].Members } ON ROWS
FROM TestCube
WHERE ( [Measures].[Packages] )
```

得到的值表格类似于下表，表中在 COLUMNS 和 ROWS 轴维度的各个交集显示 Packages 度量值的值。

	Air	**sea**
1st quarter	60	50
2nd quarter	45	45

MDX 首先评估轴维度和切片器维度，并在从要查询的多维数据集检索信息之前生成结果多维数据集的结构。

指定轴维度的内容

轴维度决定多维结果集的边缘。多维表达式 (MDX) 使用 SELECT 子句通过将集合指派到特定轴来指定轴维度。以下信息描述在 MDX 中怎样处理这种指派。

在下面的语法示例中，每个 <axis_specification> 值定义一个轴维度。数据集中轴的个数等于多维表达式 (MDX) 查询中 <axis_specification> 值的个数。MDX 查询最多可以支持 128 个指定轴，但几乎没有 MDX 查询会用到 5 个以上的轴。

<axis_specification> 语法可分解为：

<axis_specification> ::= <set> ON <axis_name>

<axis_name> ::= COLUMNS | ROWS

轴维度上的只能接受集合<set>，如果是手工指定成员集合，必须用{}包围，如果使用 MDX 集合函数，则不需要用{}包围，因为集合函数返回值为集合。一个轴维度上可以包含几个维度，如：

SELECT {[Measures].[Sales_Dollars], [Measures].[Sales_Units], [Measures].[Sales_Units_max]} ON columns, CrossJoin({[State].[Canada], [State].[Mexico], [State].[USA]}, {[Product].[Bread], [Product].[Dairy], [Product].[Meat]}) ON rows FROM sales WHERE ([Time].[All Time], [Employee].[All Employee])

Columns 轴上是手工指定的成员元组集合，用{}包围，Rows 轴使用集合函数 CrossJoin，该函数返回两个集合的交集，Rows 轴上包含两个维度 State 和 Product。

指定切片器维度的内容

切片器维度筛选多维数据。可以通过将切片器维度包含在多维表达式（MDX）查询的 WHERE 子句来限制所返回的数据。

假定未显式指派给轴的维度是切片器维度，并用其默认成员进行筛选。则默认成员为最高级别的第一个成员。

切片器维度还可通过使用 MDX 语法的 WHERE 子句进行显式指定。WHERE 子句的语法可分解为：

[WHERE [<slicer_specification>]]

切片器维度只可接受评估为单个元组的表达式。如下例所示：

WHERE ([Time].[1st half], [Route].[nonground])

3.3 高级查询 - 使用计算成员

基本 MDX 查询提供了对多维数据的简单查询，MDX 查询还提供了更为丰富强大的多维查询工具。

计算成员允许在查询表达式中定义公式，并将该公式当成一个新的成员，挂在某个维度下。它并非通过检索数据来解析，而是通过计算 MDX 表达式来返回值。在 MDX 查询中构造和使用计算成员的能力为多维数据提供了大量操作功能。多数分析功能都可以通过计算成员来实现。

MDX 通过 With 语句来创建计算成员，下面的语法用于将 WITH 关键字添加到 MDX SELECT 语句：

[WITH <formula_specification>]

[, <formula_specification>]

SELECT [<axis_specification>

[, <axis_specification>...]]

FROM [<cube_specification>]

[WHERE [<slicer_specification>]]

计算成员的 <formula_specification> 值在以下语法定义中进一步分解：

<formula_specification> ::= MEMBER <member_name>

AS '<value_expression>'

[,Format_String = <unsigned string>]

[,<cell_property>=<value_expression>...]

<member_name> 值是计算成员的完全合法名称，其中包括了该计算成员所关联的维度或级别，而 <value_expression> 值在经过计算后，将返回计算成员的值。

通过在 Format_String 中指定计算成员的格式化串，跟 Cube 定义中的 Measure 定义的 formatString 一样，采用 java.text.DecimalFormater 中的格式化语法。

例如，下面的 MDX 查询示例定义了两个计算成员。第一个计算成员 [Measures].[StoreType] 用于表示 Store Type 成员属性。第二个计算成员 [Measures].[ProfitPct] 用于计算给定商店的总利润率，并将其表示为格式化的百分比值。

```
WITH
    MEMBER [Measures].[StoreType] AS
    '[Store].CurrentMember.Properties("Store Type")',          其意不详
    SOLVE_ORDER = 2
    MEMBER [Measures].[ProfitPct] AS
    '(Measures.[Store Sales] - Measures.[Store Cost]) / Measures.[Store Sales]),
```

```
SELECT
    { [Store].[Store Name].Members} ON COLUMNS,
    { [Measures].[Store Sales], [Measures].[Store Cost], [Measures].[StoreType],
    [Measures].[ProfitPct] } ON ROWS
FROM Sales
```

计算成员可在层次结构的任意位置创建。例如，下列 MDX 查询示例定义作为 [Beer and Wine] 成员的子成员创建的计算成员 ,以确定给定商店的啤酒和果酒（Beer and Wine）的单位销售额是否至少为 100.00：

```
WITH
    MEMBER [Product].[Beer and Wine].[BigSeller] AS
  'IIf([Product].[Beer and Wine] > 100, "Yes","No")'
SELECT
    {[Product].[BigSeller]} ON COLUMNS,
    {Store.[Store Name].Members} ON ROWS
FROM Sales
```

也可创建不仅取决于多维数据集中的现有成员，而且取决于同一 MDX 表达式中定义的其它计算成员的计算成员。下列示例说明这样的 MDX 表达式：

```
WITH
    MEMBER [Measures].[ProfitPct] AS
    'Val((Measures.[Store Sales] - Measures.[Store Cost]) / Measures.[Store Sales])',
    FORMAT_STRING = '#.00%'
    MEMBER [Measures].[ProfitValue] AS
```

```
'[Measures].[Store Sales] * [Measures].[ProfitPct]',
 FORMAT_STRING = '#,###.##'
SELECT
{ [Store].[Store Name].Members} ON COLUMNS,
{ [Measures].[Store Sales], [Measures].[Store Cost], [Measures].[ProfitValue],
[Measures].[ProfitPct] } ON ROWS
FROM Sales
```

第二个计算成员 [Measures].[ProfitValue] 使用第一个计算成员 [Measures].[ProfitPct] 中创建的值来生成自己的值。

只能是后面的计算成员使用前面的计算成员，不能反过来，所有，计算成员的顺序很重要。

3.4 高级查询 - 使用命名集合

命名集合是一个与别名相关联的集合表达式。它定义一个集合，可以在 MDX 语句中多出使用，主要是为了方便维护，提高可读性，如果该集合被多处使用，可以重用，减少解析时间，提高性能。命名集别名视为集合表达式，并可用于任何接受集合表达式的地方。

下面的语法用于将 WITH 关键字添加到 MDX SELECT 语句：

```
[WITH <formula_specification>]
        [, <formula_specification>]
SELECT [<axis_specification>
         [, <axis_specification>...]]
   FROM [<cube_specification>]
[WHERE [<slicer_specification>]]
```

命名集的 <formula_specification> 值进一步分解为以下语法定义：

<formula_specification> ::= SET <set_name> AS '<set>'

<set_name> 参数包含命名集的别名。<set> 参数包含命名集别名所指的集合表达式。

例如，[ChardonnayChablis] 命名集用于 Product 维度上的所有 Chardonnay 酒和 Chablis 酒成员。以下示例对命名集的语法进行了描述：

```
WITH SET [ChardonnayChablis] AS
    '{[Product].[All Products].[Drink].[Alcoholic Beverages].[Beer and Wine].[Wine].[Good].[Good Chardonnay],
    [Product].[All Products].[Drink].[Alcoholic Beverages].[Beer and Wine].[Wine].[Pearl].[Pearl Chardonnay],
    [Product].[All Products].[Drink].[Alcoholic Beverages].[Beer and Wine].[Wine].[Portsmouth].[Portsmouth Chardonnay],
    [Product].[All Products].[Drink].[Alcoholic Beverages].[Beer and Wine].[Wine].[Top Measure].[Top Measure Chardonnay],
    [Product].[All Products].[Drink].[Alcoholic Beverages].[Beer and Wine].[Wine].[Walrus].[Walrus Chardonnay],
    [Product].[All Products].[Drink].[Alcoholic Beverages].[Beer and Wine].[Wine].[Good].[Good Chablis Wine],
    [Product].[All Products].[Drink].[Alcoholic Beverages].[Beer and Wine].[Wine].[Pearl].[Pearl Chablis Wine],
    [Product].[All Products].[Drink].[Alcoholic Beverages].[Beer and
```

Wine].[Wine].[Portsmouth].[Portsmouth Chablis Wine],

[Product].[All Products].[Drink].[Alcoholic Beverages].[Beer and Wine].[Wine].[Top Measure].[Top Measure Chablis Wine],

[Product].[All Products].[Drink].[Alcoholic Beverages].[Beer and Wine].[Wine].[Walrus].[Walrus Chablis Wine]}'

还可在用于创建命名集的集合表达式中使用 MDX 函数。

4. MDX 函数列表

4.1 维度函数

表示返回值为 Dimension 的函数.

4.1.1 Dimension

返回包含指定的成员、级别或层次结构的维度。

语法

成员

«Member».**Dimension**

返回包含 «Member» 的维度。

级别

«Level».**Dimension**

返回包含 «Level» 的维度。

层次结构

«Hierarchy».**Dimension**

返回包含 «Hierarchy» 的维度。

4.1.2 Dimensions

返回由数值表达式或字符串表达式指定的维度。

语法

数字

Dimensions(«Numeric Expression»)

返回这样的维度，该维度在多维数据集中以零为基的位置是由 «Numeric Expression» 指定的。

说明 Measures 维度总由 Dimensions(0) 表示。

字符串

Dimensions(«String Expression»)

返回这样的维度，该维度的名称是由 «String Expression» 指定的。

注释

Dimensions 函数的字符串版本通常用于用户定义的函数。

示例

如果将 Time、Region 和 Product 维度添加到多维数据集（按照所列顺序），则下面的表达式将返回 Region：

Dimensions(2)

4.2 层级函数

返回值为 Hierarchy 的函数。

注：实际上，在本文档中，我们比较少介绍 Hierarchy 的概念，因为我们建议一个维度只能有一个 Hierarchy，这时，Hierarchy 基本上等同于 Dimension。因此，用户可以不用层级函数。

4.2.1 Hierarchy

返回包含指定的成员或级别的层次结构。

语法

成员

«Member».**Hierarchy**

返回包含 «Member» 的层次结构。

级别

«Level».**Hierarchy**

返回包含 «Level» 的层次结构。

示例

成员

[January].Hierarchy

该示例返回 Time。

级别

[Quarter].Hierarchy

该示例返回 Time。

4.3 级别函数

4.3.1 Level

返回成员的级别。

语法

«Member».**Level**

示例

如果 Time 维度有"（全部）"、Year、Quarter、Month、Week 和 Day 级别，则以下示例返回 Month 级别：

January.Level

下面的示例返回 Month 级别的名称：

January.Level.Name

4.3.2 Levels

返回由数值表达式或字符串表达式指定的级别。

语法

数字

«Dimension».**Levels**(«Numeric Expression»)

返回其基于零的位置是由 «Numeric Expression» 指定的级别。

字符串

Levels(«String Expression»)

返回其名称是由 «String Expression» 指定的级别。

注释

字符串版本的 **Levels** 函数用于用户定义的函数。

示例

下面的示例假定 Time 维度有"(全部)"、Year、Quarter、Month、Week 和 Day 级别。

数字

以下示例返回 Quarter 级别：

```
Time.Levels(2)
```

字符串

以下示例返回 Year 级别：

```
Levels("Year")
```

4.4 成员函数

4.4.1 Ancestor

返回指定级别或指定距离上成员的祖先。

语法

级别

Ancestor(«Member», «Level»)

返回 «Level» 中指定的维度级别中的 «Member» 的祖先。

距离

Ancestor(«Member», «Numeric Expression»)

返回层次结构中与 «Member» 的距离为 «Numeric Expression» 个步骤的祖先。

示例

如果 Geography 维度包括级别 Country、State 和 City，下列函数将返回如下值。

表达式	返回
Ancestor(Los Angeles, Country)	[USA]
Ancestor(Los Angeles, State)	[California]
Ancestor(Los Angeles, 0)	[Los Angeles]
Ancestor(Los Angeles, 1)	[California]
Ancestor(Los Angeles, 2)	[USA]

4.4.2 ClosingPeriod

返回成员在指定级别上的后代中的最后一个兄弟。

语法

ClosingPeriod([«Level»[, «Member»]])

注释

如果指定 «Level»，则使用包含 «Level» 的维度，否则使用 Time 维度。如果没有指定 «Level»，则使用 «Member» 所在级别的下一级别。如果未指定 «Level» 或 «Member»，则默认设置为 Time.**CurrentMember**。

此函数等同于 **BottomCount(Descendants(**«Member»**,** «Level»**), 1)**。

OpeningPeriod 函数与此函数类似，但返回第一个兄弟而不是最后一个兄弟。

示例

下例返回 [1991].December：

```
ClosingPeriod(Month, [1991])
```

4.4.3 Cousin

返回父成员下方与指定子成员具有相同的相对位置的子成员。

语法

Cousin(«Member1», «Member2»)

注释

此函数在级别内的成员顺序和位置上操作。如果有两个维度 ,第一个有四个级别，第二个有五个级别，则第一个维度第三级别的同代是第二个维度的第三级别。

示例

在下面的示例中，假定年份 1996 和 1994 在成员 March 之前都包含相同数目的月份：

Cousin([1996].March, [1994])

本例产生成员 [1994].March。

如果同一示例假定 1996 级别包含 January、February、March、April、May、June、July、August、September、October、November 和 December 成员 ,而 1994 级别包含 [1st Quarter]、[2nd Quarter]、[3rd Quarter] 和 [4th Quarter] 成员，则该示例返回 [1994].[3rd Quarter]，因为它处于该级别内的同一相对位置（第三）。

4.4.4 CurrentMember

返回迭代过程中维度上的当前成员。

语法

«Dimension».**CurrentMember**

注释

在维度成员集合的迭代过程中，迭代过程的每一步中正在被操作的成员就是当前成员。此函数返回该成员。

示例

Time..CurrentMember

4.4.5 DefaultMember

返回维度或层次结构的默认成员。

语法

维度

«Dimension».**DefaultMember**

层次结构

«Hierarchy».**DefaultMember**

注释

如果维度包含"（全部）"级别，则默认成员是"全部"成员。如果该属性是空的而维度或层次结构不包含"（全部）"级别，则默认成员是最高级别的任何成员。后一种情况下，**DefaultMember** 函数不明确，我们处理为计算后物理存储的第一个成员。

示例

维度

如果 Time 维度包含级别"(全部)"、Year、Quarter、Month 和成员 All-Time，则下面的表达式返回 All-Time：

Time.DefaultMember

层次结构

如果 [Fiscal Year] 层次结构包含级别 Quarter、Month 和成员 [Month 1]，则下列表达式返回 [Month 1]：

[Fiscal Year].DefaultMember

4.4.6 FirstChild

返回成员的第一个子代。

语法

«Member».**FirstChild**

示例

如果 Time 维度包含级别 Year、Quarter、Month、Week 和 Day，则下面的代码返回 January：

[1995].FirstChild

4.4.7 FirstSibling

返回成员的父代的第一个子代。

语法

«Member».**FirstSibling**

示例

假定维度由月份组成，则下面的示例返回 January：

May.FirstSibling

4.4.8 Lag

返回成员，该成员处在其所在维度中指定的成员之前指定的位数上。

语法

«Member».**Lag**(«Numeric Expression»)

注释

成员在维度中的位置由维度的自然顺序决定。位置的编号以零为基。

如果 «Numeric Expression» 为零，则返回 «Member»。如果 «Numeric Expression» 为负数，则返回其后的成员。

Lag(1) 等同于 **PrevMember**。**Lag**(-1) 等同于 **NextMember**。

«Member».**Lead** 函数与此类似，但是方向相反。**Lag**(*n*) 等同于 **Lead**(-*n*)。

示例

如果 Time 维度中的级别包括 Year 和 Month，则下面的示例返回 [1994].November：

[1995].February.Lag(3)

4.4.9 LastChild

返回指定成员的最后一个子代。

语法

«Member».**LastChild**

示例

如果 Time 维度包含 Year、Quarter、Month、Week 和 Day，则下面的示例返回 December：

1995.LastChild

4.4.10 LastSibling

返回指定成员的父代的最后一个子代。

语法

«Member».**LastSibling**

示例

如果父代级别为 quarters，则以下示例返回 June：

May.LastSibling

3.29 lastPeriods

Syntax:

lastPeriods (integer_exp , member)

Description:

Returns the set of members from the same level that ends with the specified member. The number of members returned is the absolute value of "integer_exp". If “integer_exp" is negative then members following and including the specified member are returned. Typically used with a time dimension.

Example:

lastPeriods(2,[2006 Q 4])

result: 2006 Q 3

2006 Q 4

lastPeriods(-3,[2006 Q 4])

result: 2006 Q 2

2006 Q 3

2006 Q 4

4.4.11 Lead

返回成员，该成员处在其所在维度中指定的成员后指定的位数上。

语法

«Member».**Lead**(«Numeric Expression»)

注释

成员在维度中的位置由维度的自然顺序决定。位置的编号以零为基。

如果 «Numeric Expression» 为零，则返回 «Member»。如果 «Numeric Expression» 为负数，则返回前面的成员。

Lead(1) 等同于 **NextMember**。**Lead**(-1) 等同于 **PrevMember**。

«Member»**.Lag** 函数与此类似，但方向相反。**Lead**(*n*) 等同于 **Lag**(-*n*)。

示例

如果 Time 维度中的级别包括 Year 和 Month，则下面的示例返回 [1995].February：

```
[1994].November.Lead(3)
```

4.4.12 NextMember

返回指定成员所在级别的下一个成员。

语法

«Member».**NextMember**

注释

NextMember 函数返回与 «Member» 中所指定的成员位于同一级别的下一个成员。

示例

如果 Year 级别是由名为 [1994]、[1995] 和 [1996] 的成员组成，则下例返回 [1995]：

[1994].NextMember

4.4.13 OpeningPeriod

返回某一指定级别的后代中的第一个兄弟 ,或者某一指定成员的后代中的第一个兄弟。

语法

OpeningPeriod([«Level»[, «Member»]])

注释

如果指定 «Level»，则使用包含 «Level» 的维度，否则使用 Time 维度。如果没有指定 «Level»，则使用 «Member» 所在级别的下一级别。如果未指定 «Level» 和 «Member»，则默认设置为 Time.**CurrentMember**。

此函数等同于 **TopCount(Descendants(**«Member»**,** «Level»**)**, 1**)**。

ClosingPeriod 函数与此类似，但返回最后一个兄弟而非第一个兄弟。

示例

下例返回 [1991].January：

```
OpeningPeriod(Month, [1991])
```

4.4.14 ParallelPeriod

返回上一时期中与指定成员具有相同的相对位置的成员。

语法

ParallelPeriod([«Level»[, «Numeric Expression»[, «Member»]]])

注释

此函数类似于 **Cousin** 函数，但与 Time 系列的关系更密切。它在 «Level»（称作 *ancestor*）中提取 «Member» 的祖先，然后提取滞后 «Numeric Expression» 的 *ancestor* 的兄弟，并返回那个兄弟后代中 «Member» 的并行时期。

此函数有如下默认值：

- 如果没有指定 «Level»，则 «Member» 值默认为 **Time.CurrentMember**。否则就是 *dimension*.**CurrentMember**，其中 *dimension* 是 *level* 所属的维度。
- «Numeric Expression» 默认为 1。
- «Level» 默认为 «Member» 的父代级别。

此函数等同于 **Cousin(**Member**,Lag(Ancestor(**Member**,**Level**),**Numeric Expression**)**。

示例

下表列出如何使用此函数的各种示例。

表达式	返回
ParallelPeriod(Year,2, [96 Qtr 3])	[94 Qtr 3]
ParallelPeriod(Year,2)	Time.**CurrentMember** 两年前的并行时期。 即，如果 Time.**CurrentMember** 是 [1993June]，则返回的成员是 [1991June]。
ParallelPeriod(Year)	Time.**CurrentMember** 一年前的并行时期。 即，如果 Time.**CurrentMember** 是 [1993June]，则

	返回的成员是 [1992June]。
ParallelPeriod()	**Time.CurrentMember** 父代的前一个兄弟中的并行时期。 例如，如果 **Time.CurrentMember** 是 [1993June]，则返回的成员是 [1993March]。[1993June] 的父代是 Quarter2 ,它的前一个兄弟是 Quarter1 ,Quarter1 中的并行时期是 [1993March]。

4.4.15 Parent

返回成员的父代。

语法

«Member».**Parent**

注释

此函数返回在 «Member» 中指定的成员的父成员。

示例

如果 Geography 维度包括 State 和 Country，则以下示例返回 USA：

California.Parent

4.4.16 PrevMember

返回指定成员所在级别的上一个成员。

语法

«Member».**PrevMember**

注释

此函数返回与 «Member» 中所指定的成员位于同一级别的前一个成员。

示例

如果 Year 级别是由 [1994]、[1995] 和 [1996] 组成的，则下例返回成员 [1995]：

[1996].PrevMember

4.4.17 StrToMember

返回具有多维表达式（MDX）格式的字符串表达式中的成员。

语法

StrToMember(«String Expression»)

返回具有 MDX 格式的字符串中的成员，该字符串应包含有成员，并在 «String Expression» 中指定。

注释

StrToMember 函数通常用于用户定义的函数。

示例

下例返回成员 Time.[1996]：

```
StrToMember("Time.[1996]")
```

4.5 集合函数

返回集合的函数。

4.5.1 Ancestors

返回指定级别或距离的成员的所有祖先的集合。

语法

级别

Ancestors(«Member», «Level»)

返回 «Level» 中指定级别的 «Member» 的所有祖先。

返回成员的集合必须全部来自同一层次结构，但是 «Level» 不必是与 «Member» 同一层次结构的级别。

距离

Ancestors(«Member», «Numeric Expression»)

返回层次结构中 «Member» 之上 «Numeric Expression» 个步骤的层次结构的所有成员。**Ancestors** 函数的这一形式适用于父代级别未知或无法命名的情况。返回成员的集合必须都来自同一层次结构。

说明 **Ancestors**(«Member», 0) 返回 «Member»。

注释

与 **Ancestor** 函数不同，**Ancestors** 是一个集合值表达式；它返回的是一个集合而不是一个成员。

示例

如果 Geography 维度包括级别 Country、State 和 City，下列函数将返回如下值。

表达式	返回
Ancestors([Los Angeles], Country)	{ USA }
Ancestors([Los Angeles], State)	{ California }
Ancestors([Los Angeles], 0)	{ [Los Angeles] }
Ancestors([Los Angeles], 1)	{ California }
Ancestors([Los Angeles], 2)	{ USA }

4.5.2 Ascendants

返回指定成员的祖先集合。

语法

Ascendants(«Member»)

注释

Ancestor 函数返回特定级别上的一个特定的祖先成员或祖先，而 **Ascendants** 函数则不同，它对成员的层次结构执行后序遍历，并且在集合中返回与给定成员相关的所有祖先成员，包括其自身。

示例

下例返回集合 { LA, California, USA, [All Locations] }：

Ascendants([Los Angeles])

4.5.3 BottomCount

从集合底端开始返回指定数目的项，可以选择首先对集合排序。

语法

BottomCount(«Set», «Count»[, «Numeric Expression»])

注释

此函数根据 «Numeric Expression» 的值对集合进行排序，然后返回底端 «Count» 成员，其中的 «Count» 是一个数值表达式。

重要 与 **TopCount** 函数相似，**BottomCount** 函数总是打破层次结构。

示例

BottomCount(Geography.Cities.Members, 5, Sales)

4.5.4 BottomPercent

对集合排序并返回指定数目的最底部的元素 ,这些元素的累积合计至少为指定的百分比。

语法

BottomPercent(«Set», «Percentage», «Numeric Expression»)

注释

此函数对在 «Set« 中指定的集合进行排序，并返回其 «Numeric Expression» 的累积合计至少为 «Percentage» 的指定数量的最底端元素。«Percentage» 是一个数值表达式。

重要 与 **TopPercent** 函数相似，**BottomPercent** 函数总是打破层次结构。

示例

BottomPercent(Products.[Product Description].Members, 25, Sales)

4.5.5 BottomSum

使用数值表达式对集合排序，并返回指定数量的最底端元素，这些元素的和至少为指定的值。

语法

BottomSum(«Set», «Value», «Numeric Expression»)

注释

此函数按 «Numeric Expression« 进行排序并挑选最底端的 *n*（可能的最小数）个元素，这些元素的和至少是 «Value»。

重要 与 **TopSum** 函数相似，**BottomSum** 函数总是打破层次结构。

示例

BottomSum(Products.[Product Description].Members, 100000, Quantity)

4.5.6 Children

返回成员的子代。

语法

«Member».**Children**

示例

下列代码返回 { Nebraska, Oklahoma, Montana } :

[Geography].[All Geography].[Central Region].Children

4.5.7 Crossjoin

返回两个集合的矢量积。

语法

Crossjoin(«Set1», «Set2»)

替代语法

«Set1» * «Set2»

注释

所得集合中元组的顺序取决于 «Set1» 和 «Set2» 的顺序 ,以及其成员的顺序。

如果 «Set1» = {*x1*, *x2*,...,*xn*} 并且 «Set2» = {*y1*, *y2*, ..., *yn*} , 则 **Crossjoin**(*Set1*, *Set2*) 为 :

{(*x1*, *y1*), (*x1*, *y2*),...,(*x1*, *yn*), (*x2*, *y1*), (*x2*, *y2*),...,

(*x2*, *yn*),..., (*xn*, *y1*), (*xn*, *y2*),..., (*xn*, *yn*)}

示例

下例返回 {([1994], USA), ([1994], Japan), ([1995], USA), ([1995], Japan)}：

```
CrossJoin({[1994], [1995]}, {USA, Japan})
```

4.5.8 Descendants

返回某一成员在指定级别或距离上的后代集合，可以选择包含或不包含其它级别上的后代。

语法

级别

Descendants(«Member», [«Level»[, «Desc_flags»]])

返回成员的后代集合，该成员由 «Member» 指定，位于 «Level» 所指定的级别，并且可以选择由 «Desc_flags» 中指定的标志进行修改。

如果没有指定 «Level» 或 «Desc-flags» 参数，则函数的语法如下：

Descendants(«Member», «Member».Level, SELF_BEFORE_AND_AFTER)

距离

Descendants(«Member», «Distance»[, «Desc_flags»])

返回成员的后代，而该成员由 «Member»（在层次结构中有 «Distance» 个步骤）指定，并且可以选择由 «Desc_flags» 中指定的标志进行修改。此语法通

常用于处理不齐整层次结构。指定 «Distance» 为 0 将返回一个只包含 «Member» 中指定的成员的集合。

标志

标志	描述
SELF	默认值。只从 «Level» 中返回后代成员。当且仅当指定的 «Level» 是 «Member» 的级别时，才包括 «Member»。
AFTER	从所有从属于 «Level» 的级别中返回后代成员。
BEFORE	返回 «Member» 和 «Level» 之间所有级别的后代成员 ,不包括来自 «Level» 的成员。
BEFORE_AND_AFTER	从所有从属于 «Member» 的级别的所有级别中返回后代成员，不包括来自 «Level» 中的成员。
SELF_AND_AFTER	返回 «Level» 中和 «Level» 的所有从属级别中的后代成员。
SELF_AND_BEFORE	返回 «Level» 中和 «Member» 与 «Level» 之间所有级别中的后代成员。
SELF_BEFORE_AFTER	返回所有从属于 «Member» 级别的所有级别的后代成员。
LEAVES	返回在 «Member» 及 «Level» 或 «Distance» 之间的叶后代成员。

注释

默认情况下，只包含指定级别或距离上的成员。此函数与 SELF 的 «desc_flag» 值相对应。通过更改 «desc_flags» 的值，可以包含或不包含位于指定级别或距离的后代、指定级别或距离之前或之后（直到叶节点为止）的子代、以及与指定的级别或距离无关的所有叶子代。

示例

假定 Location 维度中级别的命名为（按照层次结构顺序）Countries（国家）、States（州）、Counties（县）和 Cities（城市）。

表达式	返回
Descendants(USA)	美国 (USA) 的所有州 (State)、县 (County) 和城市 (City)
Descendants(USA, Counties)	美国 (USA) 的所有县 (County)
Descendants(USA, Counties, SELF)	美国 (USA) 的所有县 (County)
Descendants(USA, Counties, BEFORE)	美国 (USA) 的所有州 (State)
Descendants(USA, Counties, AFTER)	美国 (USA) 的所有城市 (City)
Descendants(USA, Counties, BEFORE_AND_AFTER)	美国 (USA) 的所有州 (State) 和城市 (City)
Descendants(USA, Counties, SELF_BEFORE_AFTER)	美国 (USA) 的所有州 (State)、县 (County) 和城市 (City)
Descendants(USA, States, LEAVES)	美国 (USA) 的所有州 (State) 以及国家

	(Country) 级别和州 (State) 级别间的任何叶成员。
Descendants(USA, 1)	美国 (USA) 的所有州 (State)
Descendants(USA, 2, SELF_BEFORE_AFTER)	美国 (USA) 的所有州 (State)、县 (County) 和城市 (City)

4.5.9 Distinct

返回一个集合，并从指定集合中删除重复的元组。

语法

Distinct(«Set»)

注释

从尾部删除重复的元组。

示例

下例返回 {(a,b), (c,d)}：

Distinct({(a,b), (c,d), (a,b)})

4.5.10 Except

查找两个集合之间不同的项，可以选择保留重复项。

语法

Except(«Set1», «Set2»[, ALL])

注释

在查找不同的项之前先消除两个集合中的重复项。可选的 ALL 标志保留重复项。清除 «Set1» 中的匹配重复项并保留非匹配重复项。

示例

示例

Except({Canada, [British Columbia], Mexico, [British Columbia], USA, Washington},

{Canada, Mexico, California})

返回

{[British Columbia], USA, Washington}

示例

Except({Canada, [British Columbia], Mexico, [British Columbia], USA, Washington},

{Canada, Mexico, California}, ALL)

返回

{[British Columbia], [British Columbia], USA, Washington}.

4.5.11 Filter

返回根据搜索条件对指定集合进行筛选所得到的集合。

语法

Filter(«Set», «Search Condition»)

注释

Filter 函数对 Set 中指定的集合的每个成员，评估多维表达式（MDX）逻辑表达式（该表达式在 Search Condition 中指定），并返回满足搜索条件的成员集合。

Filter 函数的工作方式与 **IIf** 函数的类似。**IIf** 函数仅返回两个选项中的一个，返回其中哪个取决于 MDX 逻辑表达式的取值，而 **Filter** 函数返回满足指定搜索条件的一组成员。实际上，**Filter** 函数对该集合中的每个成员执行 **IIf**(«Search Criteria», «Member», NULL)，并返回所得结果。如果所有成员都不满足搜索条件，则返回空集。

示例

如果这些城市从 1995 到 1996 级别的销售额下降，则以下示例返回 {Paris, Buffalo}：

```
Filter(SampleSet, (Sales,[1996]) < (Sales, [1995]))
```

Expression <#High Margin Product Lines#>:
filter([Product line], [Gross margin] > .30)

Revenue	<#Year#>	<#Year#>
<#High Margin Product Lines#>	<#1234#>	<#1234#>
<#High Margin Product Lines#>	<#1234#>	<#1234#>

Results:

Revenue	2004	2005	2006
Golf Equipment	$5,597,980.86	$9,598,268.88	$10,709,215.84
Mountaineering Equipment	$0.00	$9,642,674.54	$11,248,676.06
Outdoor Protection	$1,536,456.24	$988,230.64	$646,428.04
Personal Accessories	$7,144,797.52	$10,955,708.04	$13,793,960.30

Source Data:

Year	Product line	Gross margin
2006	Camping Equipment	28.18%
2006	Golf Equipment	45.58%
2006	Mountaineering Equipment	36.59%
2006	Outdoor Protection	55.50%
2006	Personal Accessories	34.13%

Expression:
filter([Product line], tuple([Gross margin], [2006]) > .30)
Results:

Revenue	2004	2005	2006
Golf Equipment	$5,597,980.86	$9,598,268.88	$10,709,215.84
Mountaineering Equipment	$0.00	$9,642,674.54	$11,248,676.06
Outdoor Protection	$1,536,456.24	$988,230.64	$646,428.04
Personal Accessories	$7,144,797.52	$10,955,708.04	$13,793,960.30

4.5.12 Generate

将集合应用到另一集合的每个成员，然后用 union 运算合并所得的集合。或者，返回一个在某个集合上计算字符串表达式所得的值创建的串联字符串。

语法

集合

Generate(«Set1», «Set2»[, ALL])

字符串

Generate(«Set», «String Expression»[, «Delimiter»])

注释

此函数的集合版本将 «Set2» 应用于 «Set1» 的每个成员，然后用 union 运算合并所得的集合。如果指定了 ALL，结果中将保留重复项。

此函数的字符串版本迭代于 «Set» 中指定的集合的每个成员，对成员计算字符串表达式（在 «String Expression» 中指定）的值，然后将结果串联成返回字符串。或者，可以用 «Delimiter» 提供字符串表达式（即用分隔符来分隔串联返回字符串中的每个结果），从而对字符串进行分隔。

示例

集合

Generate({USA, France}, Descendants(Geography.CurrentMember, Cities))

此函数对集合 {USA, France} 中的每个成员应用表达式 **Descendants**(Geography.**CurrentMember**, Cities)。每应用于一个成员便产生一个集合。（应用于 USA 将生成美国 (USA) 所有城市的集合；应用于 France 将生成法国 (France) 所有城市的集合。）这些集合通过 union 运算合并以返回此函数的结果。本例中，美国和法国的所有城市就是结果。通常，**Generate**(«Set1», «set_expression») 将把 «set_expression» 应用于 «Set1» 的每个成员，然后用 union 运算合并所得的结果。

如果通过 **CurrentMember**，«Set1» 与 «set_expression» 无关，那么 **Generate** 生成 «set_expression» 所指的集合的简单复制，它包含的复制与 «Set1» 中的元组一样多。如果指定了可选的 ALL 标志，结果中将保留所有重复项。如果未指定 ALL，重复项将被删除。例如：

Generate({USA, FRANCE}, {SEATTLE, BOSTON}, ALL)

返回集合

{SEATTLE, BOSTON, SEATTLE, BOSTON}.

但如果未指定 ALL，那么返回的集合是

{SEATTLE, BOSTON}.

字符串

以下示例返回字符串 "19971998"：

Generate({Time.[1997], Time.[1998]}, Time.CurrentMember.Name)

以下示例返回字符串 "1997 and 1998"：

Generate({Time.[1997], Time.[1998]}, Time.CurrentMember.Name, " and ")

4.5.13 Head

返回集合中指定数目的前若干个元素。

语法

Head(«Set»[, «Numeric Expression»])

注释

此函数返回集合中前 «Numeric Expression» 个元素。保留元素的顺序。«Numeric Expression» 的默认值是 1。如果 «Numeric Expression» 小于 1，则返回空集合。如果 «Numeric Expression» 超出了集合中元组的个数，则返回原始的集合。

示例

以下示例返回集合 {USA, Canada, France}：

Head({USA, Canada, France, Germany, Japan}, 3)

4.5.14 Hierarchize

在层次结构中对集合的成员排序。

语法

Hierarchize(«Set»[, POST])

注释

此函数在层次结构中对 «Set» 的成员排序。除非使用 POST 关键字，否则当未指定其它排序条件时，级别中的成员按其自然顺序排序，这是维度上的成员的默认排序顺序。POST 关键字使用后采用后序排序，即以后序方式遍历层级树。

Hierarchize 始终保留重复项。

示例

示例

Hierarchize(SampleSet)

按自然顺序返回集合。按层次结构排列的数据集如下所示（假定数据源的自然顺序是按字母排列）：

France		
		Nice
		Paris
UK		
		London
USA		
	California	
		LA
		Buffalo
		NYC

4.5.15 Intersect

返回两个输入集合的交集，可以选择保留重复项。

语法

Intersect(«Set1», «Set2»[, ALL])

注释

此函数返回 «Set1» 和 «Set2» 的交集。根据默认设置，在相交之前先删除两个集合中的重复项。

可选的 ALL 保留重复项。ALL 有几种工作方式。算法是：不重复的元素照常相交。对于 «Set1» 中的每个重复项，将其与 «Set2» 中的重复项相匹配，如果存在匹配的项，则在交集中保留匹配的重复项。

示例

示例

Intersect({[1994], [1995], [1996]}, {[1995], [1996], [1997]})

返回集合 {[1995], [1996]}。

4.5.16 Members

返回维度、级别或层次结构中成员的集合。

语法

维度

«Dimension».**Members**

此语法返回 «Dimension» 中所有成员的集合。

层次结构

«Hierarchy».**Members**

此语法返回 «Hierarchy» 中所有成员的集合。

级别

«Level».**Members**

该语法返回维度中指定级别上所有成员的集合。

示例

维度

示例

Geography.Members

返回 Geography 维度中所有成员的集合。

层次结构

示例

Time.Quarterly.Members

返回 Time 维度的 Quarters 层次结构中所有成员的集合。

级别

如果 Year 级别包含 [1994]、[1995] 和 [1996]，则该示例返回集合 {[1994], [1995], [1996]}：

Year.Members

4.5.17 Mtd

返回 Time 维度 Month 级别上的成员集合，从第一个时期开始到指定的成员为止。

语法

Mtd([«Member»])

注释

Mtd 函数是 **PeriodsToDate** 函数的快捷函数，它将函数的 «Level» 参数定义为 Month。如果未指定成员，则根据默认设置是 **Time.CurrentMember**。

Mtd(«Member») 等同于 **PeriodsToDate**(Month, «Member»)。

示例

下例将返回 1997 年 9 月的头五天。

MTD([05-Sep-1997])

4.5.18NonEmptyCrossjoin

以一个集合的形式返回两个或多个集合的矢量积 ,不包括空元组以及无相关事实数据表数据的元组。

语法

NonEmptyCrossjoin(«Set1», «Set2»[, «Set3»...][, «Crossjoin Set Count»])

注释

NonEmptyCrossjoin 函数以一个集合的形式返回两个或多个集合的矢量积，不包括空元组或无基础事实数据表提供的数据的元组 ,因此所有计算成员均被自动排除。如果未指定 Crossjoin Set Count，则所有指定的集合将交叉联接，而且从所得集合中排除空成员。如果已指定 «Crossjoin Set Count»，则交叉联接以 «Set1» 开头的 «Crossjoin Set Count»。剩余集合用于确定在所得交叉联接集合中哪些成员是非空的。

例如，您要查看参加 Big Time Savings 促销的 Beverly Hills 每个商店的单位总销售额，但仅想查看位于 California 的那些客户的总销售额。但是，下列多维表达式 (MDX) 语法返回一个集合，该集合包含在 California 所有城市的单位总销售额，是按 Beverly Hills 的商店对客户进行分组的，而且都参加了 Big Time Savings 促销；该集合是三个集合的矢量积。所返回的单位销售额仅是三

个集合的矢量积；未返回参加 Big Time Savings 促销的 Beverly Hills 的商店的总销售额，而仅是参加促销本身的每个商店或客户城市的各个单位销售额。

```
NonEmptyCrossJoin([Store].[Beverly Hills].Children, [Customers].[CA].Children,
{[Promotions].[Big Time Savings]})
```

上面的示例范围太窄，无法完成任务。相反，下列 MDX 语句仅交叉联接前两个集合，而且从返回的集合中删除了非空成员。因为未使用 {[Promotions].[Big Time Savings]}，所以下一 MDX 语句范围太宽；MDX 语句包含太多元组，无法完成目标。

```
NonEmptyCrossJoin([Store].[Beverly Hills].Children, [Customers].[CA].Children)
```

下列 MDX 语句使用 Crossjoin Set Count 参数返回一个集合，该集合包含 California 所有城市的单位销售额，而且按 Beverly Hills 的商店对客户进行分组；该集合是前两个集合的矢量积。但是，只返回了参加 Big Time Savings 促销的交叉联接集合中的那些成员，所以完成了任务。在 Crossjoin Set Count 参数中指定的前两个集合交叉联接，而第三个集合用于在确定交叉联接集合成员是否包含数据时确定应考虑交叉联接集合的哪些成员。

```
NonEmptyCrossJoin([Store].[Beverly Hills].Children, [Customers].[CA].Children,
{[Promotions].[Big Time Savings]},2)
```

NonEmptyCrossjoin 函数的好处是处理涉及两个以上集合的交叉联接时更快、更有效，而且函数所提供的语法更简单。使用下列 MDX 语句中所示的 **Filter**、**Crossjoin** 和 **IsEmpty** 函数也可获得相同结果，但效率较低：

```
Filter(Crossjoin([Store].[Beverly Hills].Children, [Customers].[CA].Children), NOT
IsEmpty([Promotions].[Big Time Savings])
```

当添加其它集合时，使用 **Filter**、**Crossjoin** 和 **IsEmpty** 就变得不太实际，因为每个 **Crossjoin** 语句嵌套在另一个 **Crossjoin** 语句中才能返回相同结果。例如，使用 **NonEmptyCrossjoin** 将 [Product].Children 集合添加到返回的集合中类似于：

```
NonEmptyCrossJoin([Store].[Beverly Hills].Children, [Customers].[CA].Children,
[Product].Children, {[Promotions].[Big Time Savings]}, 3)
```

在另一方面，使用 **Filter**、**Crossjoin** 和 **IsEmpty** 函数执行相同功能与下列类似：

```
Filter(Crossjoin(Crossjoin([Store].[Beverly Hills].Children, [Customers].[CA].Children),
[Product].Children), NOT IsEmpty([Promotions].[Big Time Savings]))
```

前一 MDX 语句在执行时比较慢，而且易读性比其基于 **NonEmptyCrossjoin** 的对应语句要差。

示例

下列语句返回一个集合，该集合包含 California 所有城市的全部单位销售额，而且按参加 Big Time Savings 促销的 Beverly Hills 的商店对客户进行分组：

NonEmptyCrossJoin([Store].[Beverly Hills].Children, [Customers].[CA].Children,

{[Promotions].[Big Time Savings]},2)

请参见

4.5.19 Order

排列指定集合的成员，可以选择保留或打破层次结构。

语法

Order(«Set», {«String Expression» | «Numeric Expression»}

[, ASC | DESC | BASC | BDESC])

注释

Order 有两种变化形式：按层次结构排列（ASC 或 DESC）和不按层次结构排列（BASC 或 BDESC，其中 B 代表打破层次结构）。按层次结构排列的排序首先按成员在层次结构中的位置对其进行排序，然后再对每个级别进行排序。而不按层次结构排列的排序在对集合中的成员排序时不考虑层次结构。如果没有明确说明，则根据默认设置使用 ASC。

示例

示例

Order(SampleSet, ([1995], Sales), DESC)

按层次结构排列所有成员然后按 Sales 排序每个级别。在构造排序后的列表时，比较最高级别上的 Sales。因此，如果 California 的所有城市的 Sales 合计低于 New York 的所有城市的 Sales 合计，在按降序排列的列表中，California 和 California.LA 将排在 NYC 的下面。

该示例的结果

Order(SampleSet, ([1995], Sales), DESC)

在下表中列出。

位置			1995 年销售数据
USA			5000
	California		2000
		LA	500
		Buffalo	300
		NYC	900
France			2500
		Paris	365
		Nice	27
UK			1900

		London	250

下面的表达式在按成员的值排序成员时不考虑它们在成员层次结构中的相对位置。在本例中，按每个城市的 1995 年销售数据排序数值，包括按州和国家/地区计算的累积销售数据：

Order(SampleSet, ([1995], Sales), BDESC)

下表显示上一个表达式的结果。

位置	1995 年销售数据
USA	5000
France	2500
California	2000
UK	1900
NYC	900
LA	500
Paris	365
Buffalo	300
London	250
Nice	27

说明 如果输入的集合中有两个元素，这两个元素的 «String Expression» 或 «Numeric Expression» 有相同的值，则保留输入顺序。

例如，如果 USA 和 Europe 的销售额各为 300，而 Asia 的销售额为 100，则以下表达式返回的是集合 {Asia, USA, Europe}，而不是集合 {Asia, Europe, USA}：

```
Order({USA, Europe, Asia}, Sales, BASC)
```

4.5.20 PeriodsToDate

返回指定级别上的一个时期(成员)集合，从第一个时期开始到指定的成员为止。

语法

PeriodsToDate([«Level»[, «Member»]])

注释

在 «Level» 作用域内，此函数返回处于 «Member» 级别上的时期的集合，该集合以第一个时期开始，以 «Member» 结束。如果没有指定级别或成员，则 «Member» 的值就为 Time.**CurrentMember**，并且 «Level» 为 Time.**CurrentMember** 的父代级别。如果指定了级别，则 «Member» 为 *dimension*.**CurrentMember**，其中的 *dimension* 是 «Level» 的维度。

示例

下表列出可以使用 **PeriodsToDate** 的各种方法。

PeriodsToDate(Quarter, [05-Oct-2007]) 我觉得是 2007 年 9 月 1（第三季度第一天到 10 月 5 号的值）

表达式	返回
PeriodsToDate(Quarter, [05-Sep-1997])	从 Quarter3 开始的天数的集合。
PeriodsToDate(Year, March)	集合 {January, February, March}。
PeriodsToDate(Year)	从 Time.**CurrentMember** 的祖先所在的年份开始一直到 **Time.CurrentMember** 为止的成员的集合。
PeriodsToDate()	从 **Time.CurrentMember** 的时期所在的级别开始一直到 **Time.CurrentMember** 为止的成员的集合。所有返回的成员与 **Time.CurrentMember** 位于同一级别上。

PeriodsToDate(*level*, *member*) 与 **TopCount**(**Descendants**(**Ancestor**(*member*, *level*), *member*.**Level**), 1):*member* 相同

4.5.21 Qtd

返回 Time 维度 Quarter 级别上的成员集合 ,从第一个时期开始到指定的成员为止。

语法

Qtd([«Member»])

注释

PeriodsToDate 函数的这个快捷函数将 **PeriodsToDate** 函数的 «Level» 参数预定义为 Quarter。如果未指定成员，则根据默认设置是 **Time.CurrentMember**。

Qtd(«Member») 等同于 **PeriodsToDate**(Quarter, «Member»)。

示例

下例将返回从 1997 年第三季度开始的天数的集合：

```
Qtd([05-Sep-1997]))
```

4.5.22 Siblings

返回指定成员的兄弟，包括成员本身。

语法

«Member».**Siblings**

示例

下例返回集合 { January, February, March }：

[Time].[All Time].[1998].[Quarter 1].[January].Siblings

4.5.23 Subset

从指定集合中返回元组的子集。

语法

Subset(«Set», «Start»[, «Count»])

注释

此函数从 «Set» 中返回 «Count» 元组，并作为一个集合，该集合从 «Start» 位置开始。«Start» 是以零为基：0 对应于集合中的第一个元组，1 对应于第二个元组，依此类推。如果没有指定 «Count»，则返回从集合 «Start» 开始到集合末尾的所有元组。

示例

下例返回集合 {USA, Canada}：

Subset({USA, Canada, France, Germany, Japan, England, Peru}, 0, 2)

从 0 数 2 个　就是第一个和第二个

4.5.24 Tail

从集合尾部返回子集。

语法

Tail(«Set»[, «Count»])

注释

此函数返回集合中后面 «Count» 个元素。保留元素的顺序。«Count» 的默认值为 1。如果 «Count» 小于 1，则返回空集合。如果 «Count» 超出了集合中元组的个数，则返回原始的集合。

示例

以下代码返回集合 {France, Germany, Japan}：

```
Tail({USA, Canada, France, Germany, Japan}, 3)
```

4.5.25 TopCount

从指定集合的顶端成员开始，返回指定数目的项目，可以选择首先对集合排序。

语法

TopCount(«Set», «Count»[, «Numeric Expression»])

注释

此函数根据 «Numeric Expression» 的值对集合进行排序，然后返回顶端 «Count» 成员，其中的 «Count» 是一个数值表达式。

重要 与 **BottomCount** 函数相似，此函数总是打破层次结构。

示例

Topcount(Geography.Cities.Members, 5, Sales)

4.5.26 TopPercent

对集合排序并返回最顶端的元素，这些元素的累积合计至少为指定的百分比。

语法

TopPercent(«Set», «Percentage», «Numeric Expression»)

注释

此函数使用 «Numeric Expression» 对集合进行排序，并返回其 «Numeric Expression» 的累积合计至少为 «Percentage» 的顶端 *n* 个元素。

«Percentage» 是一个数值表达式。

重要 与 **BottomPercent** 函数相似，此函数总是打破层次结构。

示例

TopPercent({London, Paris, Rome, New York, Seattle, Tokyo}, 15, Sales)

4.5.27 TopSum

对集合排序并返回最顶端的元素，这些元素的累积合计至少为指定的值。

语法

TopSum(«Set», «Value», «Numeric Expression»)

注释

此函数在 «Numeric Expression» 上排序并挑选顶端的 *n*（可能的最小数）个元素，这些元素的累积合计至少是 «Value»。

重要 与 **BottomSum** 函数相似，此函数总是打破层次结构。

示例

Topsum(Products.[Product Description].Members, 100000, Quantity)

4.5.28 Union

返回对两个集合进行 union 运算所生成的集合，可以保留重复的成员。

语法

Union(«Set1», «Set2»[, ALL])

注释

此函数返回 «Set1» 和 «Set2» 的 union 运算结果，并在默认情况下消除重复项。ALL 标志表示在并集中保留重复项。从尾部删除重复项。

Union(USA.Children, CANADA.Children, ALL)

4.5.29 Wtd

返回 Time 维度 Week 级别上的成员集合 ,从第一个时期开始到指定的成员为止。

语法

Wtd([«Member»])

注释

Wtd 函数是 **PeriodsToDate** 函数的快捷函数，它将 **PeriodsToDate** 函数的 «Level» 参数定义为 Week。如果未指定成员，则默认为 Time.**CurrentMember**。

Wtd(«Member») 等同于 **PeriodsToDate**(Week, «Member»)。

示例

下面的示例将返回从一周的开始到当前日的天数：

```
Wtd(Day)
```

4.5.30 Ytd

返回 Time 维度 Year 级别上的成员集合，从第一个时期开始到指定的成员为止。

语法

Ytd([«Member»])

注释

Ytd 函数是 **PeriodsToDate** 函数的快捷函数 ,它将 **PeriodsToDate** 函数的 «Level» 参数定义为 Year。如果未指定成员，则根据默认设置是 **Time.CurrentMember**。

Ytd(«Member») 等同于 **PeriodsToDate**(Year, «Member»)。

示例

下例将返回从 **Time.CurrentMember** 祖先的起始年到 **Time.CurrentMember** 的成员集合：

Ytd()

4.5.31：

返回处于成员对之间的成员集合.

语法

<Member>:<Member>

注释

两个 Member 所属的 Level 必须相同.

示例

下例将返回从 **Time.[2000].[1]**到 **Time.[2000].[5]**之间的成员集合.

Time.[2000].[1]: Time.[2000].[5]

4.6 逻辑函数

4.6.1 IsEmpty

如果表达式的值为空单元值，则返回 TRUE，否则返回 FALSE。

语法

IsEmpty(«Value Expression»)

示例

如果 Measures.CurrentMember 是空单元，则以下示例返回 TRUE：

IsEmpty(Measures.CurrentMember)

4.6.2 And

逻辑与.

<Logical Expression> AND <Logical Expression>

4.6.3 Or

逻辑或.

<Logical Expression> OR <Logical Expression>

4.6.4 Not

逻辑非.

<Logical Expression> OR <Logical Expression>

4.6.5 Xor

逻辑异或.

<Logical Expression> XOR <Logical Expression>

4.6.6 >

比较大小.两个版本:

<Numeric Expression> > <Numeric Expression>

<String> > <String>

4.6.7 >=

比较大小.两个版本:

<Numeric Expression> >= <Numeric Expression>

<String> >= <String>

4.6.8 <

比较大小.两个版本:

<Numeric Expression> < <Numeric Expression>

<String> < <String>

4.6.9 <=

比较大小.两个版本:

<Numeric Expression> <= <Numeric Expression>

<String> <= <String>

4.6.10 <>

比较大小.两个版本:

<Numeric Expression> <> <Numeric Expression>

<String> <> <String>

4.6.11 =

比较大小.两个版本:

<Numeric Expression> = <Numeric Expression>

<String> = <String>

4.7 数值函数

4.7.1 Aggregate

返回根据成员的聚合类型，用适当的聚合函数计算所得的值。

语法

Aggregate(«Set»[, «Numeric Expression»])

示例

在下面的表达式中，先用度量值 SumSales 显示计算成员 Total，然后再用度量值 MaxSales 显示计算成员 Total。在前一种情况下，Total 是通过加法（使用 **Sum**）计算得到的。在后一种情况下，Total 是通过取最大值计算得到的。

```
WITH MEMBER Geography.Total AS   'AGGREGATE({USA, France})'
SELECT {Measures.SumSales, Measures.MaxSales} ON COLUMNS,
        {USA, France, Total} ON ROWS
FROM SalesCube
WHERE ([1998])
```

4.7.2 Avg

返回在某一集合上对数值表达式求得的平均值。

语法

Avg(«Set»[, «Numeric Expression»])

注释

此函数需要单元数的隐性计数，不包含空单元。若要强制包含空单元，应用程序必须使用 **CoalesceEmpty** 函数。

示例

如果 Sales 分别为 1000、2000 和 3000，则下面的函数返回 2000：

Avg({USA, Canada, Mexico}, Sales)

4.7.3 CoalesceEmpty

将空单元值合并为数字或字符串并返回合并后的值。

语法

数值

CoalesceEmpty(«Numeric Expression»[, «Numeric Expression»]...)

字符串

CoalesceEmpty(«String Expression»[, «String Expression»]...)

注释

此函数返回值表达式列表中的第一个（左侧）非空的值表达式。如果所有的值表达式都取值为空单元值，则返回空单元值。

所有值表达式都必须取值为数值数据类型或空单元值。或者，所有值表达式都必须取值为字符串数据类型或空单元值。

示例

数值

如果 Measures.CurrentMember 是空单元值，则下例返回 -99，否则返回 Measures.CurrentMember：

```
CoalesceEmpty(Measures.CurrentMember, -99)
```

字符串

下例在 Time.Parent.Name 是空单元值时，返回字符串 "EMPTY"，否则返回 Time.Parent.Name：

```
CoalesceEmpty(Time.Parent.Name, "EMPTY")
```

4.7.4 Count

返回集合中项目的数量（具体数目取决于集合）。

语法

维度

Dimensions.**Count**

返回多维数据集中的维度数，其中包括 [Measures] 维度。

级别

«Dimension»|«Hierarchy».Levels.**Count**

返回维度或层次结构中的级别数，包括 [All] 级别（如果适用）。

集合 - 语法 1

Count(«Set»[, **ExcludeEmpty** | **IncludeEmpty**])

返回集合中的单元数。该语法允许分别使用 **ExcludeEmpty** 或 **IncludeEmpty** 标志来排除或包含空单元。

集合 - 语法 2

«Set»**.Count**

返回集合中的单元数，其中包括空单元。

注释

若要从集合计数中排除空单元，请使用可选的 **ExcludeEmpty** 关键字。

示例

如果 Time 包含级别 Year 和 Month，Year 的成员是 1994 和 1995，那么下面的示例返回 24：

集合 - 示例 1

Count({Time.Month.Members})

集合 - 示例 2

Time.Month.Members.Count

4.7.5 IIf

返回由逻辑测试确定的两个数值或字符串值之一。

语法

数字

IIf(«Logical Expression», «Numeric Expression1», «Numeric Expression2»)

如果 «Logical Expression» 取值为 TRUE，则此函数返回 «Numeric Expression1»，否则，返回 «Numeric Expression2»。

字符串

IIf(«Logical Expression», «String Expression1», «String Expression2»)

如果 «Logical Expression» 取值为 TRUE，则此函数返回 «String Expression1»，否则，返回 «String Expression2»。

注释

只有当 «Logical Expression» 的值为零时，才认为该表达式是 FALSE。任何其它值都被解释为 TRUE。

不推荐用 **Iif** 函数基于搜索条件创建成员的集合。请改用 **Filter** 函数根据逻辑表达式评估指定集合中的每个成员，然后返回成员的子集合。

示例

数字

如果 Measures.CurrentMember 是空单元，则下面的示例返回 0，否则返回 1：

```
IIf(IsEmpty(Measures.CurrentMember), 0, 1)
```

字符串

如果 Measures.CurrentMember 是空单元，则下面的字符串返回字符串 "Yes"，否则返回字符串 "No"：

```
IIf(IsEmpty(Measures.CurrentMember), "Yes", "No")
```

4.7.6 Max

返回在某一集合上对数值表达式求得的最大值。

语法

Max(«Set»[, «Numeric Expression»])

注释

Max 函数返回数值表达式的最大值，该表达式在 «Numeric Expression» 中指定，对在 «Set» 中指定的集合进行求值。

示例

如果国家/地区的 Sales 分别是 1000、2000 和 3000，则返回 3000：

```
Max({USA, CANADA, MEXICO}, Sales)
```

4.7.7 Median

返回在某一集合上对数值表达式求得的中值。

语法

Median(«Set»[, «Numeric Expression»])

注释

Median 函数返回数值表达式的中值，该表达式在 «Numeric Expression» 中指定，对在 «Set» 中指定的集合进行求值。

示例

如果国家/地区的 Sales 分别是 1000、2000 和 3000，则下例返回 2000：

```
Median({USA, CANADA, MEXICO}, Sales)
```

4.7.8 Min

返回在某一集合上对数值表达式求得的最小值。

语法

Min(«Set»[, «Numeric Expression»])

注释

Min 函数返回数值表达式的最小值 ,该表达式在 «Numeric Expression» 中指定，对在 «Set» 中指定的集合进行求值。

示例

如果国家/地区的 Sales 分别是 1000、2000 和 3000，则返回 1000：

Min({USA, CANADA, MEXICO}, Sales)

4.7.9 Sum

返回在某一集合上对数值表达式求得的和。

语法

Sum(«Set»[, «Numeric Expression»])

示例

如果 USA、CANADA 和 MEXICO 成员的值分别为 1000、2000 和 3000，则以下示例返回 6000：

Sum({USA, CANADA, MEXICO}, Sales.VALUE)

以下示例更加直观，并且同样有效：

Sum({USA, CANADA, MEXICO}, Sales)

4.7.10 Value

返回度量值的值。

语法

«Member».**Value**

注释

Value 函数返回在 «Member» 中指定的度量值的值。它是度量值的默认属性。

示例

Sales.Measures.CurrentMember.Value

4.7.11 +

顾名思义,数字相加.

4.7.12 -

顾名思义,数字相减.

4.7.13 *

顾名思义,数字相乘.

4.7.14 /

顾名思义,数字相除.

4.8 字符串函数

4.8.1 Generate

参考集合函数中 Generate 的描述.

4.8.2 IIf

返回由逻辑测试确定的两个数值或字符串值之一。

参考数值函数中 IIF 的描述.

4.8.3 Name

返回级别、维度、成员或层次结构的名称。

语法

维度

«Dimension».**Name**

级别

«Level».**Name**

成员

«Member».**Name**

层次结构

«Hierarchy».**Name**

注释

Name 函数返回对象名称，但不是唯一名称。

示例

维度

Products.Name

级别

Products.[Product Description].Name

成员

Products.[Product Description].Widgets.Name

层次结构

Time.[Fiscal Year].Name

4.8.4 Caption

类似于 Name

4.8.5 UniqueName

返回指定级别、维度、成员或层次结构的唯一名称。

语法

维度

«Dimension».**UniqueName**

级别

«Level».**UniqueName**

成员

«Member».**UniqueName**

层次结构

«Hierarchy».**UniqueName**

注释

UniqueName 函数返回对象的唯一名称，而不是其它名称。

示例

维度

Products.UniqueName

级别

Products.[Product Description].UniqueName

成员

Products.[Product Description].Widgets.UniqueName

层次结构

Time.[Fiscal Year].UniqueName

4.8.6 Properties

返回包含成员属性值的字符串。

语法

«Member».**Properties**(«String Expression»)

注释

Properties 函数返回在 «String Expression» 中指定的成员属性的值。成员属性可以是任何标准的成员属性，如 NAME、ID、KEY 或 CAPTION，它也可以是用户定义的成员属性。

示例

在 Store 维度中，如果 Store Name 级别有一个相关的成员属性 Store Manager，则下例返回 Smith：

[Store].[All Stores].[USA].[WA].[Bellingham].[Store 2].Properties("Store Manager")

4.8.9 Format

将数字格式化成字符串.

语法

Format(<Numeric Expression>, <Format String Expression>)

注释

<Format String Expression>与度量值的 FormatString 类似,采用采用 java.util.DecimalFormater 格式串.

示例

Format(measures.[Max sales],"#,##0.00"),将度量值[Max Sales]转换成格式为”#,##0.00”的字符串.

Format(200,"#,##0.00")将返回”200.00”.

4.8.10 ||

将两个字符串相连接.

语法

<String Expression> || <String Expression>

5.分析功能应用举例

这些例子相对来说比较复杂,使用了计算成员,主要演示了常用的分析功能.

5.1 成员百分比分析

函数：CurrentMember、Parent 等；

分析各城市的销售所占全部城市的总销售额百分比。

```
WITH MEMBER Measures.[Unit Sales Percent] AS '((Store.CURRENTMEMBER,
Measures.[Unit Sales]) / (Store.CURRENTMEMBER.PARENT, Measures.[Unit
Sales])) ', FORMAT_STRING = '0.00%'

SELECT
{Measures.[Unit Sales], Measures.[Unit Sales Percent]} ON COLUMNS,
```

```
ORDER(DESCENDANTS(Store.[USA].[CA], Store.[Store City],
SELF),[Measures].[Unit Sales], ASC) ON ROWS

FROM Sales
```

5.2 重要顾客分布分析

函数：Count、Sum、Filter、Descendants 等； 分析各个省份中重要顾客的数量及他们的总购买量，"重要顾客"的定义是一个顾客的购买金额或者购买数目达到或超过一定的数值。

```
WITH

MEMBER [Measures].[Qualified Count] AS

  'COUNT(FILTER(DESCENDANTS(Customers.CURRENTMEMBER,
[Customers].[Name]), ([Measures].[Store Sales]) > 10000 OR ([Measures].[Unit
Sales]) > 10))'

MEMBER [Measures].[Qualified Sales] AS
'SUM(FILTER(DESCENDANTS(Customers.CURRENTMEMBER,
```

[Customers].[Name]), ([Measures].[Store Sales]) > 10000 OR ([Measures].[Unit Sales]) > 10), ([Measures].[Store Sales]))'

SELECT

{[Measures].[Qualified Count], [Measures].[Qualified Sales]} ON COLUMNS,

DESCENDANTS([Customers].[All Customers], [State Province], SELF_AND_BEFORE) ON ROWS

FROM Sales

5.3 排序

函数：Order 对各个产品类别按照 Store Sales 指标降序排列，排序分为维内排序/整体排序。 select {[Measures].[Unit Sales], [Measures].[Store Sales]} on columns,

Order([Product].[Product Department].members, [Measures].[Store Sales], DESC) on rows from Sales

5.4 历史相关的累计值

函数：YTD、Sum、Descendants 求销售额的本年累计值 YTD()，类似还可以求解历史累计 YTD()、本月累计 MTD()、本周累计 WTD()等，以及更通用的函数 PeriodToDate()。with

member [Measures].[Accumulated Sales] as 'Sum(YTD(),[Measures].[Store

Sales])'

select {[Measures].[Store Sales],[Measures].[Accumulated Sales]} on columns,

{Descendants([Time].[1997],[Time].[Month])} on rows

from [Warehouse and Sales]

5.5 四则运算

函数：四则运算函数；在成员上及指标上均可以进行四则运算，动态派生出新的成员及指标。

WITH MEMBER MEASURES.ProfitPercent AS '([Measures].[Store Sales]-[Measures].[Store Cost])/([Measures].[Store Cost])',FORMAT_STRING = '#.00%' MEMBER [Time].[First Half 97] AS '[Time].[1997].[Q1] + [Time].[1997].[Q2]'MEMBER [Time].[Second Half 97] AS '[Time].[1997].[Q3] + [Time].[1997].[Q4]'

SELECT {[Time].[First Half 97], [Time].[Second Half 97], [Time].[1997].CHILDREN} ON COLUMNS,{[Store].[Store Country].[USA].CHILDREN} ON ROWS

FROM [Sales] WHERE (MEASURES.ProfitPercent)

5.6 逻辑判断

函数：IIf 逻辑判断可以根据不同的条件产生不同的结果。下例判断各商店是否是啤酒及白酒的大卖家。

WITH MEMBER [Product].[BigSeller] AS 'IIf([Product].[Drink].[Alcoholic Beverages].[Beer and Wine] > 100, "Yes","No")'

SELECT {[Product].[BigSeller],[Product].children} ON COLUMNS,

{[Store].[All Stores].[USA].[CA].children} ON ROWS FROM Sales

5.7 成员属性

函数：Properties、成员属性是与成员绑定的，其对应关系导致很难选择合适的使用方式。以下是使用成员属性的例子，它对应每个商店成员列出了商店类型属性，相应的，商店经理、商店规模、商店地址等属性也可以被列出。该用法稍加灵活应用就可以解决过去遇到的企业名称——〉企业代码对应展示问题。

WITH MEMBER [Measures].[StoreType] AS '[Store].CurrentMember.Properties("Store Type")',MEMBER [Measures].[ProfitPct] AS '(Measures.[Store Sales] - Measures.[Store Cost]) / Measures.[Store Sales]', FORMAT_STRING = '##.00%'

SELECT { Descendants([Store].[USA], [Store].[Store Name])} ON COLUMNS, {[Measures].[Store Sales], [Measures].[Store Cost], [Measures].[StoreType], [Measures].[ProfitPct] } ON ROWS" FROM Sales

5.8 多步计算实现复杂逻辑

函数：其实可以是任意函数合乎逻辑的组合求出从来没有买过乳制品的顾客，求解过程

是先求出每位顾客在过去购买的乳制品的数量累计 ,然后找出累计值为 0 的顾客。同样，过去遇到的求税额大于平均税额的海关的问题可以类似求出。

with member [Measures].[Dairy ever] as 'sum([Time].members, ([Measures].[Unit Sales],[Product].[Food].[Dairy]))' set [Customers who never bought dairy] as 'filter([Customers].members, [Measures].[Dairy ever] = 0)' select {[Measures].[Unit Sales], [Measures].[Dairy ever]} on columns,[Customers who never bought dairy] on rows from Sales

5.9 同期、前期

函数：PrevMember、ParellelPeriod 求解各产品销售额的去年同期值，年增长率。

with member [Measures].[Store Sales Last Period] as '([Measures].[Store Sales], Time.PrevMember)', format_string='#,###.00' member [Measures].[Yearly Increase Rate] as '([Measures].[Store Sales] - [Measures].[Store Sales Last Period])/ [Measures].[Store Sales Last Period]', FORMAT_STRING = '0.00%' select {[Measures].[Store Sales], [Measures].[Store Sales Last Period]} on columns, { [Product].members} on rows
from Sales where ([Time].[1998])

另一个例子，使用 ParellelPeriod 函数。

WITH MEMBER [Measures].[YTD Unit Sales] AS 'COALESCEEMPTY(SUM(YTD(),

[Measures].[Unit Sales]), 0)' MEMBER [Measures].[Previous YTD Unit Sales] AS '(Measures.[YTD Unit Sales], PARALLELPERIOD([Time].[Year]))' MEMBER [Measures].[YTD Growth] AS '[Measures].[YTD Unit Sales] - ([Measures].[Previous YTD Unit Sales])' SELECT {[Time].[1998]} ON COLUMNS, {[Measures].[YTD Unit Sales], [Measures].[Previous YTD Unit Sales], [Measures].[YTD Growth]} ON ROWS FROM Sales;

5.10 **Top N 分析**

函数：TopCount

求解 1998 年总购买量处于前 5 名的顾客；

select {[Measures].[Store Sales]} on columns,

{TopCount([Customers].[Customer Name].members,5, [Measures].[Store Sales])}

on rows

from Sales

where ([time].[1998])

5.11 成员过滤

函数：Filter、Except 求解 1998 年所有顾客中购买总额得到 1 万元以上的顾客，列出满足条件的顾客的名字、年购买数量、年购买金额。

Select {[measures].[Store Sales],[measures].[unit sales]} on columns,

FILTER(Customers.[Name].Members,[Measures].[Store Sales] > 10000) on rows

From sales

Whare ([time].[1998])

另外一种成员过滤（从所有的媒体类型中剔除 No Media 类型），确切的说应该是集合运算。

select {[Measures].[Unit Sales]} on columns,

except([Promotion Media].[Media Type].members,{[Promotion Media].[Media Type].[No Media]}) on rows

from Sales

5.12 时间段

函数：sum、"："运算符

求美国的商店在指定时间段内的销售额。

WITH MEMBER [Time].[1997].[Six Month] AS 'SUM([Time].[1]:[Time].[6])'

MEMBER [Time].[1997].[Nine Month] AS 'SUM([Time].[1]:[Time].[9])'

SELECT {[Time].[1997].[Six Month],[Time].[1997].[Nine Month]} ON

```
COLUMNS,{[measures].[store salse]} ON ROWS

FROM Sales

Where ([Store].[USA])
```