IBM DB2 10.1
for Linux, UNIX, and Windows

XQuery 参考

更新时间 2013 年 1 月

S151-1774-01

IBM DB2 10.1
for Linux, UNIX, and Windows

XQuery 参考

更新时间 2013 年 1 月

S151-1774-01

注意

使用此信息及其支持的产品前，请先阅读第 211 页的附录 B，『声明』下的常规信息。

修订版声明

此文档包含 IBM 的所有权信息。它在许可协议中提供，且受版权法的保护。本出版物中包含的信息不包括对任何产品的保证，且提供的任何语句都不需要如此解释。

您可在线或通过当地的 IBM 代表处订购 IBM 出版物。

- 要在线订购出版物，请转至 IBM 出版物中心，网址为: http://www.ibm.com/shop/publications/order
- 要查找当地的 IBM 代表处，请转至 IBM 全球联系人目录，网址为: http://www.ibm.com/planetwide/

要从美国或加拿大的 DB2 市场和销售部订购 DB2 出版物，请致电 1-800-IBM-4YOU（426-4968）。

您发送信息给 IBM 后，即授予 IBM 非独占权限，IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

目录

关于本书

XQuery 引用描述 DB2® 数据库用于与 XML 数据配合使用的 XQuery 语言。

它包括有关 XQuery 概念、数据类型、语言元素、XQuery 定义的函数和 DB2 的内置函数的信息。该引用还包括有关 DB2 XQuery 存在大小和 XQuery 数据类型方面的限制的信息。

第 1 章 DB2 XQuery 概念

下列主题介绍基本 XQuery 概念，并描述 XQuery 如何与 DB2 数据库配合工作。

XQuery 简介

XQuery 是万维网联盟（W3C）设计的一种函数编程语言，用于满足查询和修改 XML 数据的特定需求。

与可预测的并具有常规结构的关系数据不同，XML 数据可变性很高。XML 数据通常是不可预测、稀疏和自描述的。

由于 XML 数据的结构不可预测，所以需要对 XML 数据执行的查询通常与典型的关系查询不同。XQuery 语言提供了执行这些类型操作所需的灵活性。例如，可能需要使用 XQuery 语言执行下列操作:

- 搜索层次结构中某些未知层对象的 XML 数据。
- 对数据执行结构变换（例如，您可能想倒转层次结构）。
- 返回具有混合类型的结果。
- 更新现有 XML 数据。

XQuery 查询的组成部分

在 XQuery 中，表达式是查询的主要构建块。表达式可以进行嵌套，用来组成查询的主体。查询还可以在此主体前面具有序言。*序言*包含一系列用于定义查询处理环境的声明。*查询主体*包含用于定义查询结果的表达式。此表达式可由使用运算符或关键字组合而成的多个 XQuery 表达式构成。

第 2 页的图 1 说明了典型查询的结构。在此示例中，序言包含两个声明: 一个是版本声明，它指定要用来处理查询的 XQuery 语法的版本; 另一个是缺省名称空间声明，它指定要用于无前缀元素名称和类型名的名称空间 URI。查询主体包含一个用于构造 `price_list` 元素的表达式。`price_list` 元素的内容是将 `product` 元素按价格降序排列获得的列表。

图 1. XQuery 中的典型查询的结构

XQuery 与 SQL 的比较

DB2 支持将格式良好的 XML 数据存储在表列中，以及使用 SQL 和/或 XQuery 来从数据库中检索 XML 数据。支持将这两种语言用作主要查询语言，并且这两种语言都具有调用其他语言的功能。

XQuery

直接调用 XQuery 的查询以关键字 XQUERY 开头。此关键字指示正在使用 XQuery，因此 DB2 服务器必须使用适用于 XQuery 语言且区分大小写的规则。错误处理是根据用来处理 XQuery 表达式的接口来进行的。报告 XQuery 错误时会提供 SQLCODE 和 SQLSTATE，与报告 SQL 错误的方式相同。处理 XQuery 表达式时不会返回警告。XQuery 通过调用从 DB2 表和视图中抽取 XML 数据的函数来获取数据。也可从 SQL 查询中调用 XQuery。在此情况下，SQL 查询可通过绑定变量的方式将 XML 数据传递给 XQuery。XQuery 支持用来处理 XML 数据和构造新 XML 对象（例如，元素和属性）的各种表达式。用于 XQuery 的编程接口所提供的功能与 SQL 的功能类似，可用于预编译查询和检索查询结果。

SQL SQL 能够定义和实例化 XML 数据类型的值。包含格式良好的 XML 文档的字符串可解析为 XML 值，（可选）针对 XML 模式进行验证，在表中插入或更新。或者，可使用 SQL 构造函数来构造 XML 值；在此过程中会将其他关系数据转换为 XML 值。还提供了一些函数，使得可使用 XQuery 来查询 XML 数据以及将 XML 数据转换为关系表以用于 SQL 查询。除了可将 XML 值序列化为字符串数据以外，还可以在 SQL 数据类型与 XML 数据类型之间进行转换。

SQL/XML 提供了下列函数和谓词，以便从 SQL 来调用 XQuery:

XMLQUERY

XMLQUERY 是一个标量函数，它将 XQuery 表达式作为自变量，返回的是一个 XML 序列。该函数有一些可选参数，可用来将 SQL 值作为

XQuery 变量传递给 XQuery 表达式。可以在 SQL 查询的上下文中进一步处理由 XMLQUERY 返回的 XML 值。

XMLTABLE

XMLTABLE 是一个表函数，它使用 XQuery 表达式来从 XML 数据生成 SQL 表，SQL 可以进一步处理该 SQL 表。

XMLEXISTS

XMLEXISTS 是一个 SQL 谓词，它确定 XQuery 表达式是否返回由一个或多个项组成的序列（而不是一个空序列）。

使用 XQuery 函数检索 DB2 数据

在 XQuery 中，查询可以调用下列函数之一来获取 DB2 数据库中的输入 XML 数据: db2-fn:sqlquery 和 db2-fn:xmlcolumn。

db2-fn:xmlcolumn 函数将检索整个 XML 列，而 db2-fn:sqlquery 将检索基于 SQL 全查询的 XML 值。

db2-fn:xmlcolumn

db2-fn:xmlcolumn 函数采用一个字符串文字自变量，用于标识一个表或视图中的 XML 列，并返回该列中的 XML 值序列。此函数的自变量是区分大小写的。字符串文字自变量必须是类型为 XML 的限定列名。此函数允许您抽取整个 XML 数据列而不应用搜索条件。

在以下示例中，查询使用 db2-fn:xmlcolumn 函数来获取 BUSINESS.ORDERS 表的 PURCHASE_ORDER 列中的所有采购订单。然后，查询将对此输入数据执行操作，从这些采购订单的交货地址中抽取城市。查询结果是交付订单的所有城市的列表。

```
db2-fn:xmlcolumn('BUSINESS.ORDERS.PURCHASE_ORDER')/shipping_address/city
```

db2-fn:sqlquery

db2-fn:sqlquery 函数采用一个表示全查询的字符串自变量，并返回一个由全查询返回的 XML 值并置而成的 XML 序列。全查询必须指定单列结果集，而且列的数据类型必须为 XML。通过指定全查询，将允许您使用 SQL 的功能来向 XQuery 提供 XML 数据。该函数支持使用参数向 SQL 语句传递值。

在以下示例中，BUSINESS.ORDERS 表中包含名为 PURCHASE_ORDER 的 XML 列。在该示例中，查询使用 db2-fn:sqlquery 函数来调用 SQL，以获取交货日期为 2005 年 6 月 15 日的所有采购订单。然后，查询将对此输入数据执行操作，从这些采购订单的交货地址中抽取城市。查询结果是在 6 月 15 日交付订单的所有城市的列表。

```
db2-fn:sqlquery("
SELECT purchase_order FROM business.orders
WHERE ship_date = '2005-06-15' ")/shipping_address/city
```

要点: 由 db2-fn:sqlquery 或 db2-fn:xmlcolumn 函数返回的 XML 序列可以包含任何 XML 值（包括原子值和节点）。这些函数并不会总是返回格式良好的文档序列。例如，这些函数可能只返回单个原子值（例如，36）作为 XML 数据类型的实例。

对于名称是否区分大小写，SQL 和 XQuery 有不同的约定。当使用 db2-fn:sqlquery 和 db2-fn:xmlcolumn 函数时，您应知道这些差别。

SQL 是一种不区分大小写的语言

缺省情况下，SQL 语句中使用的所有普通标识符会自动转换为大写。因此，SQL 表和列的名称通常采用大写名称，例如，前面示例中的 BUSINESS.ORDERS 和 PURCHASE_ORDER。在 SQL 语句中，可以使用小写名称来引用这些列（例如，business.orders 和 purchase_order），在处理 SQL 语句时这些名称会自动转换为大写。（在 SQL 中，还可以通过将名称用双引号引起来创建称为*定界标识*的区分大小写的名称）。

XQuery 是一种区分大小写的语言

XQuery 不会将小写名称转换为大写名称。这种差别会在同时使用 XQuery 和 SQL 时导致混淆。传递给 db2-fn:sqlquery 的字符串被解释为 SQL 查询并由 SQL 解析器进行解析，这会将所有名称都转换为大写。因此，在 db2-fn:sqlquery 示例中，表名 business.orders 以及列名 purchase_order 和 ship_date 既可以大写形式也可以小写形式出现。但是，db2-fn:xmlcolumn 的操作数不是 SQL 查询。操作数是区分大小写的、用于表示列名的 XQuery 字符串文字。因为实际列名是 BUSINESS.ORDERS.PURCHASE_ORDER，所以在 db2-fn:xmlcolumn 的操作数中必须用大写指定此名称。

XQuery 和 XPath 数据模型

XQuery 表达式对 XQuery 和 XPath 数据模型（XDM）的实例进行运算并返回数据模型的实例。

XDM 是对一个或多个 XML 文档或片段的抽象表示。数据模型会定义 XQuery 中的表达式的允许值，包括中间计算期间使用的值。

将 XML 数据解析为 XDM，并在 XQuery 处理数据之前针对模式来验证这些数据。在生成数据模型期间，将解析输入 XML 文档，并将它转换为 XDM 的实例。在解析文档时，可以进行验证，也可以不进行验证。

XDM 是按照原子值和节点序列来进行描述的。

序列和项

XQuery 和 XPath 数据模型（XDM）的实例为序列。*序列*是 0 个项或多个项的有序集合。一个*项*就是一个原子值或一个节点。

一个序列可以包含节点、原子值或者是节点和原子值的任意组合。例如，下面列表中的每个条目都是一个序列:

- `36`
- `<dog/>`
- `(2, 3, 4)`
- `(36, <dog/>, "cat")`
- `()`

除列表中的条目之外，存储在 DB2 数据库的 XML 列中的 XML 文档是一个序列。

示例中用来表示序列的表示法，与用来构造 XQuery 中的序列的语法是一致的:

- 序列中的每项之间用逗号分隔。
- 整个序列是用圆括号括起来的。

- 一对空的圆括号表示一个空序列。
- 如果一个项在它自身上方出现，那么相当于一个只包含一项的序列。

  例如，序列 (36) 与原子值 36 没有区别。

不能对序列进行嵌套。当组合两个序列时，获得的结果始终是节点和原子值的平铺序列。例如，将序列 (2, 3) 追加至序列 (3, 5, 6) 时将生成单个序列 (3, 5, 6, 2, 3)。因为决不会出现嵌套序列，所以，组合这些序列时并不会生成 (3, 5, 6, (2, 3))。

不包含任何项的序列称为*空序列*。可使用空序列来表示缺少的信息或未知信息。

原子值

*原子值*是由 XML 模式定义的其中一种内置原子数据类型的实例。这些数据类型包括 String、Integer、Decimal、Date 和其他原子类型。这些类型都被描述为原子类型，原因是它们无法再细分了。

与节点不同的是，原子值没有标识。原子值的每个实例（例如，整数 7）与该值的其他每个实例都完全相同。

下列示例是一些生成原子值的方法:

- 通过一个称为“原子化”的过程从节点中抽取。每当需要原子值序列时，表达式就会使用原子化。
- 指定为数字或字符串文字。XQuery 会将文字解释为原子值。例如，下列文字就会被解释为原子值:
  - “this is a string”（类型为 xs:string）
  - 45（类型为 xs:integer）
  - 1.44（类型为 xs:decimal）
- 由构造函数计算获得。例如，以下构造函数将根据字符串“2005-01-01”来构建类型为 xs:date 的值:

  ```
  xs:date("2005-01-01")
  ```
- 由内置函数 fn:true() 和 fn:false() 返回。这些函数将返回布尔值 `true` 和 `false`。这些值不能表示为文字。
- 由多种表达式（例如，算术表达式和逻辑表达式）返回。

节点层次结构

组成一个或多个*层次结构*或*树*的节点序列，这些层次结构或树由一个根节点和可从该根节点直接或间接访问的所有节点组成。

每个节点只属于一个层次结构，而每个层次结构只有一个根节点。DB2 支持以下 6 种节点: 文档、元素、属性、文本、处理指示信息和注释。

以下 XML 文档 `products.xml` 包括一个根元素 `products`，该根元素又包含一些 `product` 元素。每个 `product` 元素都有一个名为 `pid`（产品标识）的属性以及一个名为 `description` 的子元素。`description` 元素包含名为 `name` 和 `price` 的子元素。

```
<products>
  <product pid="10">
    <description>
      <name>Fleece jacket</name>
```

```
      <price>19.99</price>
    </description>
  </product>
  <product pid="11">
    <description>
      <name>Nylon pants</name>
      <price>9.99</price>
    </description>
  </product>
</products>
```

图 2 显示 products.xml 的数据模型的简化表示。该图中包括文档节点（D）、元素节点（E）、属性节点（A）和文本节点（T）。

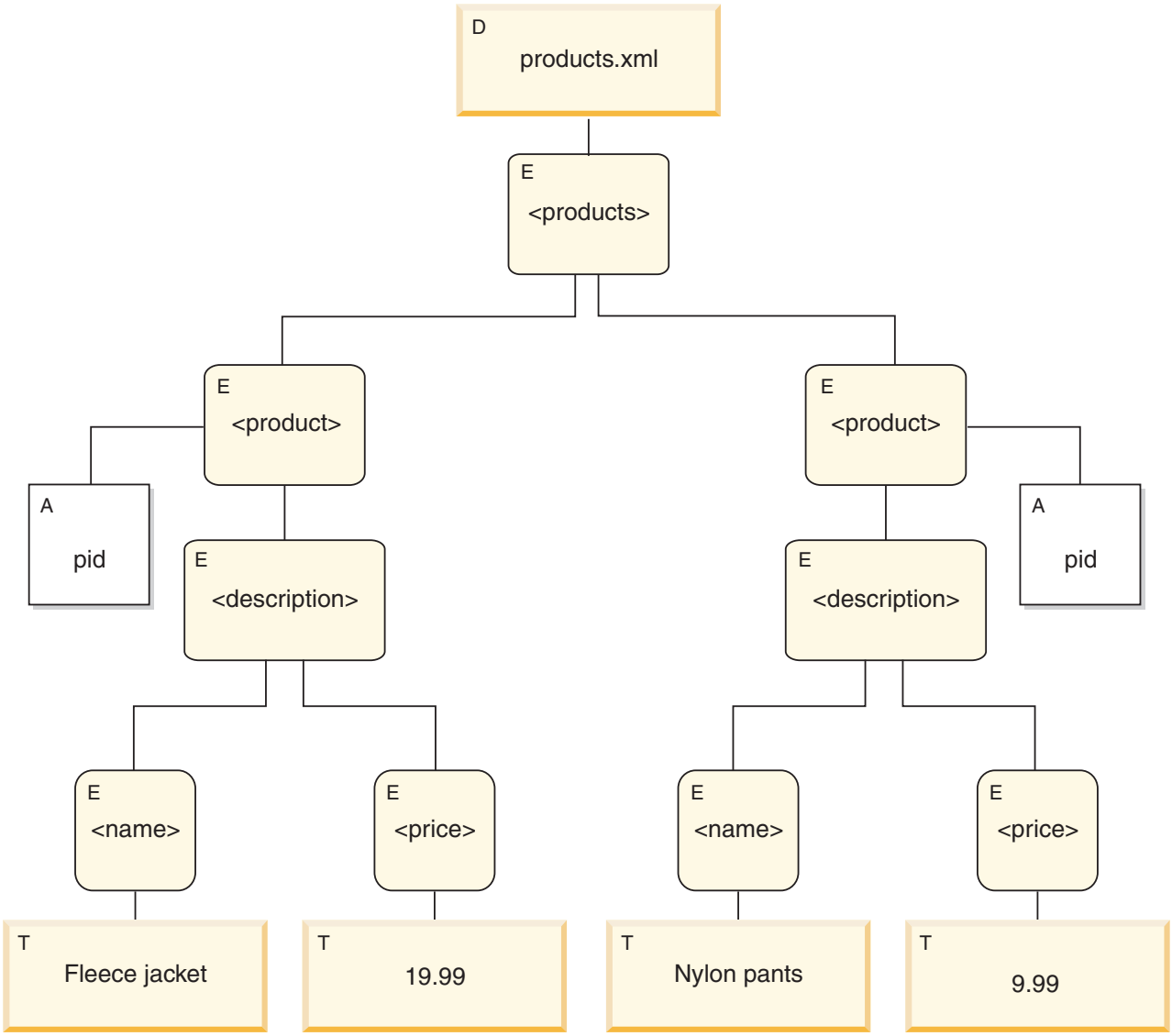


图 2. products.xml 文档的数据模型图

如示例中所示，一个节点可将其他节点作为子代，从而组成一个或多个*节点层次结构*。在该示例中，product 元素是 products 的子代。description 元素却是 product 的子代。name 和 price 元素都是 description 元素的子代。值为 Fleece Jacket 的文本节点是 name 元素的子代，而文本节点 19.99 是 price 元素的子代。

节点属性

每个节点都具有一些*特性*，这些特性用来描述该节点的特征。例如，节点的特性可能包括：节点的名称、子代、父代、属性以及用来描述该节点的其他信息。节点种类确定为特定节点提供了哪些属性。

一个节点可以具有下面的一个或多个属性:

node-name
节点的名称，表示为 QName。

父代 是当前节点的父代的节点。

type-name
节点的动态（运行时）类型（也称为*类型注释*）。

子代 是当前节点的子代的节点序列。

属性 属于当前节点的一组属性节点。

string-value
可以从节点中抽取的字符串值。

typed-value
可以从节点中抽取的由零个或零个以上原子值组成的序列。

名称空间作用域
与节点相关联的作用域内名称空间。

content
节点的内容。

节点种类

DB2 支持以下 6 种节点：文档、元素、属性、文本、处理指示信息和注释。

文档节点

文档节点包含 XML 文档。

文档节点可以具有多个子代，也可以没有子代。子代可以是元素节点、处理指令节点、注释节点和文本节点。

文档节点的字符串值等于将它的所有后代文本节点的内容按文档顺序进行并置的结果。字符串值的类型为 xs:string。文档节点的类型值与字符串值相同，但类型值的类型为 xdt:untypedAtomic。

文档节点具有下列节点属性:

- children（可能是空的）
- string-value
- typed-value

可以在 XQuery 表达式中使用经过计算获得的构造函数来构造文档节点。db2-fn:xmlcolumn 函数还可以返回一系列文档节点。

元素节点

元素节点包含 XML 元素。

一个元素可以具有一个父代，也可以没有父代；同时，它可以具有多个子代，也可以没有子代。子代可以是元素节点、处理指令节点、注释节点和文本节点。文档节点和属性节点决不会是元素节点的子代。但是，可以认为元素节点是它自己的属性的父代。元素节点的属性必须具有唯一的 QName。

元素节点具有下列节点属性:

- node-name
- parent（可能是空的）
- type-name
- children（可能是空的）
- attributes（可能是空的）
- string-value
- typed-value
- in-scope-namespaces

可以在 XQuery 表达式中使用直接构造函数或者经过计算获得的构造函数来构造元素节点。

元素节点的 type-name 属性指示它的类型值与字符串值之间的关系。例如，如果一个元素节点具有 type-name 属性 xs:decimal 和字符串值“47.5”，那么类型值就是小数值 47.5。如果元素节点的 type-name 属性是 xdt:untyped，那么元素的类型值等于其字符串值，并且类型为 xdt:untypedAtomic。

属性节点

属性节点表示 XML 属性。

属性节点可以具有一个父代，也可以没有父代。可将拥有属性的元素节点认为是它的父代，尽管属性节点不是它的父元素的子代。

属性节点具有下列节点属性:

- node-name
- parent（可能是空的）
- type-name
- string-value
- typed-value

可以在 XQuery 表达式中使用直接构造函数或者经过计算获得的构造函数来构造属性节点。

属性节点的 type-name 属性指示它的类型值与字符串值之间的关系。例如，如果一个属性节点具有 type-name 属性 xs:decimal 和字符串值“47.5”，那么它的类型值就是小数值 47.5。

文本节点

文本节点包含 XML 字符内容。

文本节点可以具有一个父代，也可以没有父代。作为文档节点或元素节点的子代的文本节点决不会作为相邻同代出现。当构造文档节点或元素节点时，任何相邻的文本节点同代都会被合并成单个文本节点。如果获得的文本节点是空的，那么会将它废弃。

文本节点具有下列节点属性:

- content（可能是空的）
- parent（可能是空的）

可以在 XQuery 表达式中使用经过计算获得的构造函数来构造文本节点，也可以通过直接元素构造函数的操作来构造文本节点。

处理指令节点

处理指令节点会封装 XML 处理指令。

处理指令节点可以具有一个父代，也可以没有父代。处理指令的内容不能包含字符串 ?>。处理指令的目标必须是一个 NCName。该目标用来标识要将指示信息发送给的应用程序。

处理指令节点具有下列节点属性:

- target
- content
- parent（可能是空的）

可以在 XQuery 表达式中使用直接构造函数或者经过计算获得的构造函数来构造处理指令节点。

注释节点

注释节点包含 XML 注释。

注释节点可以具有一个父代，也可以没有父代。注释节点的内容不能包括字符串“--”（两个连字符），并且最后一个字符不能是连字符（-）。

注释节点具有下列节点属性:

- content
- parent（可能是空的）

可以在 XQuery 表达式中使用直接构造函数或者经过计算获得的构造函数来构造注释节点。

节点的文档顺序

一个层次结构中的所有节点都要遵从某一顺序（即，*文档顺序*）。按照该顺序，每个节点都将在其子代前面出现。如果节点层次结构是用已序列化的 XML 表示的，那么文档顺序与节点的出现顺序相对应。

层次结构中的节点按以下顺序出现:

- 根节点是第一个节点。
- 元素节点在它们的子代前面出现。

- 属性节点紧跟在与它们相关联的元素节点后面出现。属性节点的相对顺序可以是任意的，但是在处理查询期间此顺序不会改变。
- 同代的相对顺序由它们在节点层次结构中的顺序来确定。
- 一个节点的子代和后代将在该节点后面的同代前面出现。

节点标识

每个节点都有一个唯一标识。即使两个节点的名称和值都相同，也可以将它们区分开。然而，原子值没有标识。

节点标识与 ID-type 属性不相同。XML 文档中的元素可由文档作者给定 ID-type 属性。然而，节点标识是由系统自动为每个节点指定的，用户无法直接看见节点标识。

节点标识用于处理下列类型的表达式:

- 节点比较。**is** 运算符使用节点标识来确定两个节点是否具有相同标识。
- 路径表达式。路径表达式使用节点标识来消除重复的节点。
- 序列表达式。**union**、**intersect** 或 **except** 运算符使用节点标识来消除重复的节点。

节点的类型值和字符串值

每个节点都同时有*类型值*和*字符串值*。这两个节点属性用于某些 XQuery 操作（例如，原子化）和函数（例如，fn:data、fn:string 和 fn:deep-equal）的定义中。

表 1. 节点的字符串值和类型值

节点种类	字符串值	类型值
文档	xs:string 数据类型的实例，它是将它的所有后代文本节点的内容按文档顺序进行并置的结果。	xdt:untypedAtomic 数据类型的实例，它是将它的所有后代文本节点的内容按文档顺序进行并置的结果。
XML 文档中的元素	xs:string 数据类型的实例，它是将它的所有文本节点后代的内容按文档顺序进行并置的结果。	xdt:untypedAtomic 数据类型的实例，它是将它的所有文本节点后代的内容按文档顺序进行并置的结果。
XML 文档中的属性	xs:string 数据类型的实例，它表示原始 XML 文档中的属性值。	xdt:untypedAtomic 数据类型的实例，它表示原始 XML 文档中的属性值。
文本	作为 xs:string 数据类型的实例的内容。	作为 xdt:untypedAtomic 数据类型的实例的内容。
注释	作为 xs:string 数据类型的实例的内容。	作为 xs:string 数据类型的实例的内容。
处理指令	作为 xs:string 数据类型的实例的内容。	作为 xs:string 数据类型的实例的内容。

XDM 的序列化

XQuery 表达式的结果，它是 XDM 的实例，可通过*序列化*进程变换为 XML 表示法。

通过序列化，节点和原子值的序列（XDM 的实例）会转换为 XML 表示法。序列化的结果并非始终表示格式良好的文档。实际上，序列化可能生成单个原子值（例如 17）或没有公共父代的元素序列。

XQuery 没有提供函数以对 XDM 执行序列化。XDM 如何通过序列化转换为 XML 数据取决于查询的执行环境。例如，CLP（命令行处理器）会返回序列化项的序列，每个序列化项作为结果中的一行返回。例如，在 CLP 中输入查询 `XQUERY (1, 2, 3)` 会返回以下结果:

```
1
2
3
```

还可通过 SQL/XML 函数 XMLSERIALIZE 执行序列化。

XML 名称空间和 QName

*XML 名称空间*是由名称空间 URI 标识的名称集合。名称空间提供了一种方法来限定用于 XQuery 中的元素、属性、数据类型和函数的名称。使用名称空间前缀限定的名称为*限定名称*（*QName*）。

XML 名称空间可避免命名冲突。

限定名称（QName）

QName 由可选名称空间前缀和局部名组成。名称空间前缀与局部名用冒号隔开。如果名称空间前缀存在，那么会绑定至 URI（通用资源标识），并且会提供 URI 的简短形式。

查询处理期间，XQuery 会扩展 QName 并解析绑定至名称空间前缀的 URI。扩展 QName 包括名称空间 URI 和局部名。如果两个 QName 具有相同名称空间 URI 和局部名，那么表示这两个 QName 相等。这意味着即使两个 QName 的前缀不同（它们的前缀绑定至同一名称空间 URI），这两个 QName 也是匹配的。

以下示例包括 QName:

- `ns1:name`
- `ns2:name`
- `name`

在此示例中，`ns1` 是绑定至 URI `http://posample.org` 的前缀。前缀 `ns2` 将绑定至 URI `http://mycompany.com`。缺省元素名称空间是另一 URI，不同于 ns1 和 ns2 的关联 URI。所有三个元素的局部名都是 `name`。

```
<ns1:name>This text is in an element named "name" that is qualified
by the prefix "ns1".</ns1:name>

<ns2:name>This text is in an element named "name" that is qualified
by the prefix "ns2".</ns2:name>

<name>This text is in an element named "name" that is in the default
element namespace.</name>
```

此示例中的元素共用同一局部名 `name`，但不会发生命名冲突，原因是这些元素在不同名称空间中。在处理表达式期间，名称 `ns1:name` 将扩展为包括绑定至 `ns1` 的 URI 和局部名 `name` 的名称。同样，名称 `ns2:name` 将扩展为包括绑定至 `ns2` 的 URI 和局部名 `name` 的名称。因为未指定任何前缀，所以具有空前缀的元素 `name` 将绑定至缺省元素名称空间。如果某个名称使用未绑定至 URI 的前缀，那么会返回错误。

QName（限定名称）遵循 W3C Recommendation *Namespaces in XML* 中定义的语法。

静态已知名称空间

名称空间声明会将名称空间前缀绑定至 URI。用于控制 QName 在查询表达式中的解释的名称空间绑定集合称为静态已知名称空间。

静态已知名称空间是查询表达式的属性，与表达式处理的数据无关。

某些名称空间前缀是预先声明的；其他名称空间则可通过查询序言或元素构造函数中的声明进行添加。下表描述了 DB2 XQuery 包括的预先声明名称空间前缀。

表 2. DB2 XQuery 中的预先声明名称空间

前缀	URI	描述
xml	`http://www.w3.org/XML/1998/namespace`	XML 保留名称空间
xs	`http://www.w3.org/2001/XMLSchema`	XML 模式名称空间
xsi	`http://www.w3.org/2001/XMLSchema-instance`	XML 模式实例名称空间
fn	`http://www.w3.org/2005/xpath-functions`	缺省函数名称空间
xdt	`http://www.w3.org/2005/xpath-datatypes`	XQuery 类型名称空间
db2-fn	`http://www.ibm.com/xmlns/prod/db2/functions`	DB2 函数名称空间

除预先声明名称空间外，还可通过下列方式提供一组静态已知名称空间:

- 在查询序言中声明，通过使用名称空间声明或缺省名称空间声明。以下示例名称空间声明使名称空间前缀 ns1 与 URI http://mycompany.com 相关联:

  ```
  declare namespace ns1 = "http://mycompany.com";
  ```

  以下示例缺省元素/类型名称空间声明会为查询中没有前缀的元素名称设置 URI:

  ```
  declare default element namespace "http://posample.org";
  ```

- 由元素构造函数中的名称空间声明属性声明。以下示例是包含名称空间声明属性的元素构造函数，该属性将前缀 ns2 绑定至构造元素作用域内的 URI http://mycompany.com:

  ```
  <ns2:price xmlns:ns2="http://mycompany.com">14.99</ns2:price>
  ```

- 由 SQL/XML 提供。SQL/XML 可提供下列一组名称空间:
  - SQL/XML 预先声明名称空间。
  - 在 SQL/XML 构造函数和其他 SQL/XML 表达式中声明的名称空间。

  序言中的名称空间声明或元素构造函数中的后续名称空间声明属性可能覆盖 SQL/XML 提供的名称空间。元素构造函数中的名称空间声明属性可能覆盖序言中声明的名称空间。

语言约定

下列主题将描述 XQuery 语言约定。

区分大小写

XQuery 是一种区分大小写的语言。

XQuery 中的关键字使用小写字符，并且不是保留关键字。XQuery 表达式中的名称可以与语言关键字相同。

空格

大多数 XQuery 表达式中允许使用空格以改进可读性，即使空格并非表达式语法的一部分亦如此。空格由空格字符（X'20'）、回车符（X'0D'）、换行符（X'0A'）和制表符（X'09'）组成。

一般来说，空格在查询中没有意义，但在下列情况下保留空格时除外:

- 空格在字符串文字中。
- 空格通过阻止解析器将两个相邻标记识别为一个标记以使表达式变得更加明晰。
- 空格在元素构造函数中。序言中的边界空格声明确定是在元素构造函数保留还是除去空格。

例如，下列表达式需要使用空格以使表达式变得更加明晰:

- `name- name` 会产生错误。解析器将 name- 识别为单个 QName（限定名称），并在找不到运算符时返回错误。
- `name -name` 不会产生错误。解析器将第一个 `name` 识别为 QName，将减号（`-`）识别为运算符，并将第二个 `name` 识别为另一个 QName。
- `name-name` 不会产生错误。但是，表达式会解析为单个 QName，原因是连字符（`-`）在 QName 中是有效字符。
- 下列表达式都会产生错误:
  - `10 div3`
  - `10div3`

  在这三个表达式中，解析器需要空格才能单独识别每个标记。

注释

序言或查询主体中允许使用注释。注释不会影响查询处理。

注释由通过符号 `(:` 和 `:)` 定界的字符串组成。以下示例是 XQuery 中的注释:

```
(: A comment. You can use comments to make your code easier to understand. :)
```

下列通用规则适用于在 DB2 XQuery 中使用注释:

- 允许使用可忽略空格的位置都可以使用注释。*可忽略空格*就是对表达式结果没有意义的空格。
- 构造函数内容中不允许使用注释。
- 注释可相互嵌套，但每个嵌套注释必须具有左定界符和右定界符 `(:` 和 `:)`。

下列示例说明合法的注释以及会产生错误的注释:

- `(: is this a comment? ::)` 是合法注释。
- `(: is this a comment? ::) or an error? :)` 会产生错误，原因是符号 `(:` 和 `:)` 的嵌套失衡。
- `(: commenting out a (: comment :) might be confusing, but is often helpful :)` 是合法注释，原因是允许进行平衡的注释嵌套。
- `"this is just a string :)"` 是合法表达式。
- `(: "this is just a string :)" :)` 会产生错误。同样，`"this is another string (:"` 是合法表达式，而 `(: "this is another string (:" :)` 会产生错误。文字内容可能会导致注释嵌套失衡。

何处可找到有关 XQuery 的更多信息

请参阅下列资源以了解有关 DB2 XQuery 所依据规范的更多信息。

- **XQuery 1.0**

  World Wide Web Consortium. *XQuery 1.0: An XML Query Language*. W3C Recommendation, 23 January 2007. 请参阅 www.w3.org/TR/2007/REC-xquery-20070123/。

- **XQuery 1.0 and XPath 2.0 Functions and Operators**

  World Wide Web Consortium. *XQuery 1.0 and XPath 2.0 Functions and Operators*. W3C Recommendation, 23 January 2007. 请参阅 www.w3.org/TR/2007/REC-xpath-functions-20070123/。

- **XQuery 1.0 and XPath 2.0 Data Model**

  World Wide Web Consortium. *XQuery 1.0 and XPath 2.0 Data Model*. W3C Recommendation, 23 January 2007. 请参阅 www.w3.org/TR/2007/REC-xpath-datamodel-20070123/。

- **XML Query Use Cases**

  World Wide Web Consortium. *XML Query Use Cases*. W3C Working Group Note, 23 March 2007. 请参阅 www.w3.org/TR/2007/NOTE-xquery-use-cases-20070323/。

- **XML Schema**

  World Wide Web Consortium. *XML Schema, Parts 0, 1, and 2*. W3C Recommendation, 28 October 2004. 请参阅 www.w3.org/TR/2004/REC-xmlschema-0-20041028/、www.w3.org/TR/2004/REC-xmlschema-1-20041028/ 和 www.w3.org/TR/2004/REC-xmlschema-2-20041028/。

- **XML Names**

  World Wide Web Consortium. *Namespaces in XML 1.0 (Second Edition)*. W3C Recommendation, 16 August 2006. 请参阅 www.w3.org/TR/2006/REC-xml-names-20060816/。

- **Updating XML**

  World Wide Web Consortium. *XQuery Update Facility*. W3C Working Draft, 11 July 2006. 请参阅 www.w3.org/TR/2006/WD-xqupdate-20060711/。

第 2 章 类型系统

XQuery 是一种强类型化语言，在该语言中，各种表达式的操作数、运算符和函数必须符合规定的类型。DB2 XQuery 的类型系统包括 XML 模式的内置类型和 XQuery 的预定义类型。

XML 模式的内置类型在名称空间 `http://www.w3.org/2001/XMLSchema` 中，该名称空间具有预先声明的名称空间前缀 `xs`。内置模式类型的一些示例包括 xs:integer、xs:string 和 xs:date。

XQuery 的预定义类型在名称空间 `http://www.w3.org/2005/xpath-datatypes` 中，该名称空间具有预先声明的名称空间前缀 `xdt`。XQuery 的预定义类型的示例包括 xdt:untypedAtomic、xdt:yearMonthDuration 和 xdt:dayTimeDuration。

每个数据类型都有词法格式，即，可转换为给定类型或者序列化后可用于表示给定类型的值的字符串。

类型层次结构

DB2 XQuery 类型层次结构显示可在 XQuery 表达式中使用的所有类型。

第 16 页的图 3 中的层次结构包括抽象基本类型和派生类型。所有原子类型派生自数据类型 xdt:anyAtomicType。每个派生数据类型与派生它的基本类型之间会用实线连接起来。

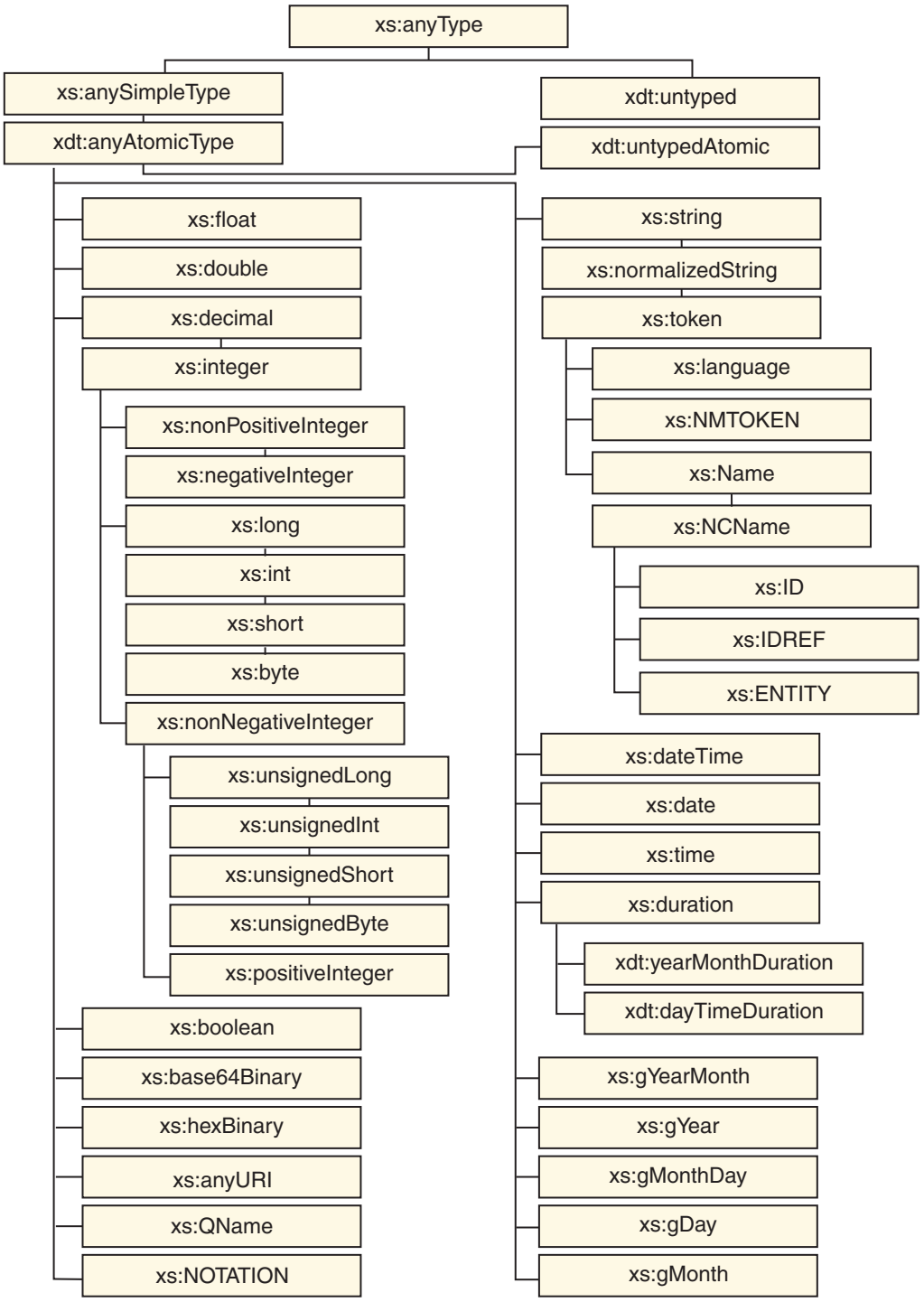


图 3. DB2 XQuery 类型层次结构

按类别划分的类型

DB2 XQuery 具有下列类型类别: 通用、无类型、字符串、数字、日期、时间、持续时间及其他。

通用数据类型

表 3. 通用数据类型

类型	描述
第 24 页的『anyType 数据类型』	xs:anyType 数据类型包含零个或零个以上节点及零个或零个以上原子值的任何序列。
第 24 页的『anySimpleType 数据类型』	xs:anySimpleType 数据类型指示可使用任何简单类型的上下文。此数据类型充当所有简单类型的基本类型。简单类型的实例可以是任何原子值序列。它派生自 xs:anyType 数据类型。
第 24 页的『anyAtomicType 数据类型』	xdt:anyAtomicType 数据类型指示可使用任何原子类型的上下文。此数据类型充当所有原子类型的基本类型。原子类型的实例是单个不可分解的值，如整数、字符串或日期。

无类型数据类型

表 4. 无类型数据类型

类型	描述
第 37 页的『untyped 数据类型』	xdt:untyped 数据类型指示未经过 XML 模式验证的节点。它派生自数据类型 xs:anyType。
第 37 页的『untypedAtomic 数据类型』	xdt:untypedAtomic 数据类型指示未经过 XML 模式验证的原子值。它派生自数据类型 xdt:anyAtomicType。

字符串数据类型

表 5. 字符串数据类型

类型	描述
第 35 页的『string 数据类型』	xs:string 数据类型表示字符串。它派生自数据类型 xdt:anyAtomicType。
第 34 页的『normalizedString 数据类型』	xs:normalizedString 数据类型表示空格规范化的字符串。它派生自数据类型 xs:string。
第 36 页的『token 数据类型』	xs:token 数据类型表示标记化字符串。它派生自 xs:normalizedString 数据类型。
第 33 页的『language 数据类型』	xs:language 数据类型表示 *RFC 3066* 定义的自然语言标识。它派生自数据类型 xs:token。
第 34 页的『NMTOKEN 数据类型』	xs:NMTOKEN 数据类型表示 *XML 1.0 (Third Edition)* 中的 NMTOKEN 属性类型。它派生自 xs:token 数据类型。
第 33 页的『Name 数据类型』	xs:Name 数据类型表示 XML 名称。它派生自 xs:token 数据类型。
第 33 页的『NCName 数据类型』	xs:NCName 数据类型表示不使用冒号的 XML 名称。它派生自 xs:Name 数据类型。
第 32 页的『ID 数据类型』	xs:ID 数据类型表示 *XML 1.0 (Third Edition)* 中的标识属性类型。它派生自 xs:NCName 数据类型。

表 5. 字符串数据类型 (续)

类型	描述
第 32 页的『IDREF 数据类型』	xs:IDREF 数据类型表示 *XML 1.0 (Third Edition)* 中的 IDREF 属性类型。它派生自 xs:NCName 数据类型。
第 30 页的『ENTITY 数据类型』	xs:ENTITY 数据类型表示 *XML 1.0 (Third Edition)* 中的 ENTITY 属性类型。它派生自 xs:NCName 数据类型。

数字数据类型

表 6. 数字数据类型

类型	描述
第 28 页的『decimal 数据类型』	xs:decimal 数据类型表示可用十进制数字表示的实数子集。它派生自数据类型 xdt:anyAtomicType。
第 30 页的『float 数据类型』	xs:float 数据类型的模式是根据 IEEE 单精度 32 位浮点类型确定的。它派生自数据类型 xdt:anyAtomicType。
第 28 页的『double 数据类型』	xs:double 数据类型的模式是根据 IEEE 双精度 64 位浮点类型确定的。它派生自数据类型 xdt:anyAtomicType。
第 33 页的『int 数据类型』	xs:int 数据类型表示小于或等于 2 147 483 647 并且大于或等于 -2 147 483 648 的整数。它派生自 xs:long 数据类型。
第 34 页的『nonPositiveInteger 数据类型』	xs:nonPositiveInteger 数据类型表示小于或等于零的整数。它派生自 xs:integer 数据类型。
第 33 页的『negativeInteger 数据类型』	xs:negativeInteger 数据类型表示小于零的整数。它派生自数据类型 xs:nonPositiveInteger。
第 34 页的『nonNegativeInteger 数据类型』	xs:nonNegativeInteger 数据类型表示大于或等于零的整数。它派生自 xs:integer 数据类型。
第 33 页的『long 数据类型』	xs:long 数据类型表示小于或等于 9 223 372 036 854 775 807 并大于或等于 -9 223 372 036 854 775 808 的整数。它派生自数据类型 xs:integer。
第 33 页的『integer 数据类型』	xs:integer 数据类型表示小于或等于 9 223 372 036 854 775 807 并大于或等于 -9 223 372 036 854 775 808 的数字。它派生自 xs:decimal 数据类型。
第 35 页的『short 数据类型』	xs:short 数据类型表示小于或等于 32767 并大于或等于 -32 768 的整数。它派生自 xs:int 数据类型。
第 25 页的『byte 数据类型』	xs:byte 数据类型表示小于或等于 127 并大于或等于 -128 的整数。它派生自 xs:short 数据类型。

表 6. 数字数据类型 (续)

类型	描述
第 36 页的『unsignedLong 数据类型』	xs:unsignedLong 数据类型表示小于或等于 9 223 372 036 854 775 807 的无符号整数。它派生自 xs:nonNegativeInteger 数据类型。
第 36 页的『unsignedInt 数据类型』	xs:unsignedInt 数据类型表示小于或等于 4 294 967 295 的无符号整数。它派生自 xs:unsignedLong 数据类型。
第 37 页的『unsignedShort 数据类型』	xs:unsignedShort 数据类型表示小于或等于 65 535 的无符号整数。它派生自 xs:unsignedInt 数据类型。
第 36 页的『unsignedByte 数据类型』	xs:unsignedByte 数据类型表示小于或等于 255 的无符号整数。它派生自 xs:unsignedShort 数据类型。
第 34 页的『positiveInteger 数据类型』	xs:positiveInteger 数据类型表示大于或等于 1 的正整数。它派生自 xs:nonNegativeInteger 数据类型。

日期、时间和持续时间数据类型

表 7. 日期、时间和持续时间数据类型

类型	描述
第 28 页的『duration 数据类型』	xs:duration 数据类型表示通过格列高利历的年、月、日、小时、分钟和秒部分表示的持续时间。它派生自数据类型 xdt:anyAtomicType。
第 37 页的『yearMonthDuration 数据类型』	xdt:yearMonthDuration 数据类型表示通过格列高利历的年和月部分表示的持续时间。它派生自 xs:duration 数据类型。
第 27 页的『dayTimeDuration 数据类型』	xdt:dayTimeDuration 数据类型表示用天数、小时数、分钟数和秒数部分表示的持续时间。它派生自 xs:duration 数据类型。
第 25 页的『dateTime 数据类型』	xs:dateTime 数据类型表示一个时刻，具有下列以整数值表示的年、月、日、小时、分钟属性、以十进制值表示的秒属性以及可选时区指示符。它派生自数据类型 xdt:anyAtomicType。
第 25 页的『date 数据类型』	xs:date 数据类型表示持续时间正好一天的时间间隔，起始于特定日期的起始时刻。xs:date 数据类型包括用整数值表示的年、月和日属性及可选时区指示符。从 xdt:anyAtomicType 数据类型派生。
第 35 页的『time 数据类型』	xs:time 数据类型表示每一天重复出现的时刻。它派生自数据类型 xdt:anyAtomicType。
第 32 页的『gYearMonth 数据类型』	xs:gYearMonth 数据类型表示特定格列高利历年份的特定格列高利历月份。格列高利历月份是在 *ISO 8601* 中定义的。它派生自数据类型 xdt:anyAtomicType。

表 7. 日期、时间和持续时间数据类型 (续)

类型	描述
第 31 页的『gYear 数据类型』	xs:gYear 数据类型表示格列高利历年份。格列高利历年份是在 *ISO 8601* 中定义的。它派生自数据类型 xdt:anyAtomicType。
第 31 页的『gMonthDay 数据类型』	xs:gMonthDay 数据类型表示重复出现的格列高利历日期。格列高利历日期是在 *ISO 8601* 中定义的。它派生自数据类型 xdt:anyAtomicType。
第 30 页的『gDay 数据类型』	xs:gDay 数据类型表示重复出现的格列高利历日期。格列高利历日期是在 *ISO 8601* 中定义的。它派生自数据类型 xdt:anyAtomicType。
第 31 页的『gMonth 数据类型』	xs:gMonth 数据类型表示每年重复出现的格列高利历月份。格列高利历月份是在 *ISO 8601* 中定义的。它派生自数据类型 xdt:anyAtomicType。

其他数据类型

表 8. 其他数据类型

类型	描述
第 25 页的『boolean 数据类型』	xs:boolean 数据类型支持二进制值逻辑的数学概念：true 或 false。它派生自数据类型 xdt:anyAtomicType。
第 24 页的『anyURI 数据类型』	xs:anyURI 数据类型表示统一资源标识 (URI)。它派生自数据类型 xdt:anyAtomicType。
第 35 页的『QName 数据类型』	xs:QName 数据类型表示 XML 限定名 (QName)。QName 包括可选名称空间前缀、用于标识 XML 名称空间的 URI 和本地部分（即 NCName）。它派生自数据类型 xdt:anyAtomicType。
第 34 页的『NOTATION 数据类型』	xs:NOTATION 数据类型表示 *XML 1.0 (Third Edition)* 中的 NOTATION 属性类型。它派生自数据类型 xdt:anyAtomicType。
第 32 页的『hexBinary 数据类型』	xs:hexBinary 数据类型表示十六进制编码的二进制数据。它派生自数据类型 xdt:anyAtomicType。
第 24 页的『base64Binary 数据类型』	xs:base64Binary 数据类型表示基本 64 位编码的二进制数据。它派生自数据类型 xdt:anyAtomicType。

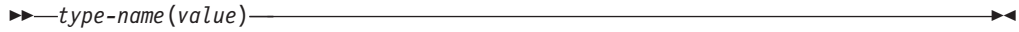
内置数据类型的构造函数

*构造函数*将一种原子类型的实例转换为另一种原子类型的实例。XML 模式中定义的每个内置原子类型都有隐式定义的构造函数。

数据类型 xdt:untypedAtomic 及两种派生数据类型 xdt:yearMonthDuration 和 xdt:dayTimeDuration 也有构造函数。

xs:NOTATION、xs:anyType、xs:anySimpleType 或 xdt:anyAtomicType 没有可用的构造函数。

内置类型的所有构造函数共用以下通用语法:



注: 构造函数 *type-name*(*value*) 的语义被定义为等价于表达式 (*value* cast as *type-name*?)。

type-name
目标数据类型的 QName。

值 要构造为目标数据类型的实例的值。将对该值应用原子化。如果原子化产生空序列，那么会返回空序列。如果原子化产生包含多项的序列，那么会产生错误。否则，生成的原子值的数据类型将转换为目标类型。有关哪些类型可转换为其他类型的信息，请参阅『类型转换』。

例如，下图表示 XML 模式数据类型 xs:unsignedInt 的构造函数的语法:

可传递至此构造函数的值是可有效转换为目标数据类型的任何原子值。例如，此函数的下列调用会返回同一结果，即 xs:unsignedInt 值 12:

```
xs:unsignedInt(12)
xs:unsignedInt("12")
```

在第一个示例中，数字文字 12 将传递至构造函数。因为文字不包含小数点，所以将解析为 xs:integer，而 xs:integer 值将转换为类型 xs:unsignedInt。在第二个示例中，字符串文字“12”将传递至构造函数。字符串文字将解析为 xs:string，而 xs:string 值将转换为类型 xs:unsignedInt。

还可将节点作为自变量调用构造函数。在此情况下，DB2 XQuery 会将节点原子化以抽取其类型值，然后使用该值调用构造函数。如果传递至构造函数的值不能转换为目标数据类型，那么会返回错误。

xs:QName 的构造函数与构造函数通用语法的不同之处在于，该构造函数被约束为将字符串文字用作自变量。

将值转换为某种数据类型时，可使用可转型表达式来测试该值是否能转换为该数据类型。

类型转换

xdt:untypedAtomic、xs:integer、xs:duration 的两种派生类型（xdt:yearMonthDuration 和 xdt:dayTimeDuration）及 XML 模式中定义的 19 种基本类型之间支持类型转换。强制类型转换表达式和类型构造函数中会使用类型转换。

下表中指示了受支持的类型转换。每个表在左边显示作为类型转换源的基本类型，并在顶部显示作为类型转换目标的基本类型。第一个表包含从 xdt:untypedAtomic 至 xs:dateTime 的目标，第二个表包含从 xs:time 至 xs:NOTATION 的目标。

表中的单元格包含下列三个字符的其中一个:

Y 是。指示支持从源类型的值转换至目标类型。

N 否。指示不支持从源类型的值转换至目标类型。

M 可能。指示进行源类型至目标类型的转换时，某些值的转换可能成功，另一些值的转换可能失败。

不支持与 xs:anySimpleType 或 xdt:anyAtomicType 之间进行转换。

如果尝试进行不受支持的强制类型转换，那么会返回错误。

表 9. 基本类型转换，第 1 部分（从 xdt:untypedAtomic 至 xs:dateTime 的目标）

源数据类型	目标 uA	目标 string	目标 float	目标 double	目标 decimal	目标 integer	目标 dur	目标 yMD	目标 dTD	目标 dT
uA	Y	Y	M	M	M	M	M	M	M	M
string	Y	Y	M	M	M	M	M	M	M	M
float	Y	Y	Y	Y	M	M	N	N	N	N
double	Y	Y	M	Y	M	M	N	N	N	N
decimal	Y	Y	Y	Y	Y	M	N	N	N	N
integer	Y	Y	Y	Y	Y	Y	N	N	N	N
dur	Y	Y	N	N	N	N	Y	Y	Y	N
yMD	Y	Y	N	N	N	N	Y	Y	N	N
dTD	Y	Y	N	N	N	N	Y	N	Y	N
dT	Y	Y	N	N	N	N	N	N	N	Y
time	Y	Y	N	N	N	N	N	N	N	N
date	Y	Y	N	N	N	N	N	N	N	Y
gYM	Y	Y	N	N	N	N	N	N	N	N
gYr	Y	Y	N	N	N	N	N	N	N	N
gMD	Y	Y	N	N	N	N	N	N	N	N
gDay	Y	Y	N	N	N	N	N	N	N	N
gMon	Y	Y	N	N	N	N	N	N	N	N
bool	Y	Y	Y	Y	Y	Y	N	N	N	N
b64	Y	Y	N	N	N	N	N	N	N	N
hxB	Y	Y	N	N	N	N	N	N	N	N
aURI	Y	Y	N	N	N	N	N	N	N	N
QN	Y	Y	N	N	N	N	N	N	N	N
NOT	Y	Y	N	N	N	N	N	N	N	N

表 10. 基本类型转换，第 2 部分（从 xs:time 至 xs:NOTATION 的目标）

源数据类型	目标 time	目标 date	目标 gYM	目标 gYr	目标 gMD	目标 gDay	目标 gMon	目标 bool	目标 b64	目标 hxB	目标 aURI	目标 QN	目标 NOT
uA	M	M	M	M	M	M	M	M	M	M	M	N	N
string	M	M	M	M	M	M	M	M	M	M	M	M	M
float	N	N	N	N	N	N	N	Y	N	N	N	N	N

表 10. 基本类型转换，第 2 部分（从 xs:time 至 xs:NOTATION 的目标）(续)

源数据类型	目标 time	目标 date	目标 gYM	目标 gYr	目标 gMD	目标 gDay	目标 gMon	目标 bool	目标 b64	目标 hxB	目标 aURI	目标 QN	目标 NOT
double	N	N	N	N	N	N	N	Y	N	N	N	N	N
decimal	N	N	N	N	N	N	N	Y	N	N	N	N	N
integer	N	N	N	N	N	N	N	Y	N	N	N	N	N
dur	N	N	N	N	N	N	N	N	N	N	N	N	N
yMD	N	N	N	N	N	N	N	N	N	N	N	N	N
dTD	N	N	N	N	N	N	N	N	N	N	N	N	N
dT	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N
time	Y	N	N	N	N	N	N	N	N	N	N	N	N
date	N	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N
gYM	N	N	Y	N	N	N	N	N	N	N	N	N	N
gYr	N	N	N	Y	N	N	N	N	N	N	N	N	N
gMD	N	N	N	N	Y	N	N	N	N	N	N	N	N
gDay	N	N	N	N	N	Y	N	N	N	N	N	N	N
gMon	N	N	N	N	N	N	Y	N	N	N	N	N	N
bool	N	N	N	N	N	N	N	Y	N	N	N	N	N
b64	N	N	N	N	N	N	N	N	Y	Y	N	N	N
hxB	N	N	N	N	N	N	N	N	Y	Y	N	N	N
aURI	N	N	N	N	N	N	N	N	N	N	Y	N	N
QN	N	N	N	N	N	N	N	N	N	N	N	N	N
NOT	N	N	N	N	N	N	N	N	N	N	N	N	M

行和列由标识下列类型的简短代码标识:

- uA = xdt:untypedAtomic
- string = xs:string
- float = xs:float
- double = xs:double
- decimal = xs:decimal
- integer = xs:integer
- dur = xs:duration
- yMD = xdt:yearMonthDuration
- dTD = xdt:dayTimeDuration
- dT = xs:dateTime
- time = xs:time
- date = xs:date
- gYM = xs:gYearMonth
- gYr = xs:gYear
- gMD = xs:gMonthDay
- gDay = xs:gDay

- gMon = xs:gMonth
- bool = xs:boolean
- b64 = xs:base64Binary
- hxB = xs:hexBinary
- aURI = xs:anyURI
- QN = xs:QName
- NOT = xs:NOTATION

anyAtomicType 数据类型

xdt:anyAtomicType 数据类型指示可使用任何原子类型的上下文。此数据类型充当所有原子类型的基本类型。原子类型的实例是单个不可分解的值，如整数、字符串或日期。它派生自 xs:anySimpleType 数据类型。

数据类型 xdt:anyAtomicType 具有无约束词法格式。

不支持在 xdt:anyAtomicType 数据类型与其他数据类型之间进行转换。

anySimpleType 数据类型

xs:anySimpleType 数据类型指示可使用任何简单类型的上下文。此数据类型充当所有简单类型的基本类型。简单类型的实例可以是任何原子值序列。它派生自 xs:anyType 数据类型。

xs:anySimpleType 数据类型具有无约束词法格式。

不支持在 xs:anySimpleType 数据类型与其他数据类型之间进行转换。

anyType 数据类型

xs:anyType 数据类型包含零个或零个以上节点及零个或零个以上原子值的任何序列。

anyURI 数据类型

xs:anyURI 数据类型表示统一资源标识 (URI)。它派生自数据类型 xdt:anyAtomicType。

xs:anyURI 数据类型的词法格式是一个合法的 URI 字符串，符合*RFC 2396*定义和*RFC 2732*修正的内容。请尽量避免在此类型的值中使用空格，否则空格将被编码为 %20。

base64Binary 数据类型

xs:base64Binary 数据类型表示基本 64 位编码的二进制数据。它派生自数据类型 xdt:anyAtomicType。

对于基本 64 位编码的二进制数据，整个二进制流是通过使用基本 64 位字母编写的。基本 64 位字母表在 *RFC 2045* 中作了描述。

xs:base64Binary 的词法格式被限制为只能使用 *RFC 2045* 中定义的基本 64 位字母表的 65 个字符。有效字符包括 a 到 z、A 到 Z、0 到 9、加号（+）、正斜杠（/）、等号（=）、*XML 1.0 (Third Edition)*中定义的字符及空格。不允许使用其他字符。

boolean 数据类型

xs:boolean 数据类型支持二进制值逻辑的数学概念：true 或 false。它派生自数据类型 xdt:anyAtomicType。

xs:boolean 数据类型的词法格式被约束为使用下列值：true、false、1 和 0。

byte 数据类型

xs:byte 数据类型表示小于或等于 127 并大于或等于 -128 的整数。它派生自 xs:short 数据类型。

xs:byte 的词法格式是一个可选符号，后跟限定长度的十进制数序列。如果符号被省略，那么表示正号（+）。下列数字是此数据类型的有效示例：`-1`、`0`、`126` 和 `+100`。

date 数据类型

xs:date 数据类型表示持续时间正好一天的时间间隔，起始于特定日期的起始时刻。xs:date 数据类型包括用整数值表示的年、月和日属性及可选时区指示符。从 xdt:anyAtomicType 数据类型派生。

类型为 xs:date 的分时区值会根据确定的时区记录一天的起始时刻。一天的起始时刻起始于 00:00:00 并延伸至 24:00:00 但不包括 24:00:00，24:00:00 是第二天的起始时刻。例如，日期 2002-10-10+13:00 的起始时刻是值 2002-10-10T00:00:00+13:00。此值等价于 2002-10-09T11:00:00Z，后者同时是 2002-10-09-11:00 的起始时刻。因此，值 2002-10-10+13:00 和 2002-10-09-11:00 表示同一时间间隔。

xs:date 的词法格式是以下格式的限定长度字符序列：*yyyy-mm-ddzzzzzz*。不允许使用负值日期。下列缩写用于描述此格式：

yyyy
: 用于表示年份的 4 位数字。有效值是 0001 到 9999 的整数。不允许使用加号（+）。

mm
: 用于表示月份的 2 位数字。

dd
: 用于表示日期的 2 位数字。

zzzzzz
: 可选。如果存在，那么表示时区。请参阅第 26 页的『时区指示符』以了解有关此属性的格式的信息。

dateTime 数据类型

xs:dateTime 数据类型表示一个时刻，具有下列以整数值表示的年、月、日、小时、分钟属性、以十进制值表示的秒属性以及可选时区指示符。它派生自数据类型 xdt:anyAtomicType。

xs:dateTime 的有效词法没有显式时区。对于没有显式时区的表示，将使用隐式时区 UTC（全球标准时间，也称为格林威治标准时间）。表示为数字值的每个属性被约束为由高一级属性确定的时间间隔内的最大值。例如，日期值不能为 32，对于 2002 年 2 月不能为 29。

xs:dateTime 的词法格式是以下格式的限定长度字符序列：*yyyy-mm-dd*T*hh*:*mm*:*ss*.*ssssssszzzzzz*。不允许使用负值日期。下列缩写用于描述此格式:

yyyy
用于表示年份的 4 位数字。有效值是 0001 到 9999 的整数。不允许使用加号（+）。

- 日期部分之间的分隔符。

mm 用于表示月份的 2 位数字。

dd 用于表示日期的 2 位数字。

T 用于指示一天中的时间的分隔符。

hh 用于表示小时的 2 位数字。仅当分钟和秒都表示为零时才允许使用值 24。包括时间 24:00:00 的查询将被视为第二天的 00:00:00 处理。

: 时间部分之间的分隔符。

mm 用于表示分钟的 2 位数字。

ss 用于表示秒的整数部分的 2 位数字。

.ssssss
可选。如果存在，那么为表示秒的小数部分的 1 到 6 位数字。

zzzzzz
可选。如果存在，那么表示时区。请参阅『时区指示符』以了解有关此属性的格式的信息。

例如，以下格式指示美国东部标准时间 2005 年 10 月 10 日中午:

```
2005-10-10T12:00:00-05:00
```

此时间用 UTC 表示为 2002-10-10T17:00:00Z。

时区指示符

时区指示符的词法格式是包括下列两种格式之一的字符串:

- 正号（+）或负号（-），后跟 hh:mm，其中使用了下列缩写:
  - *hh* 用于表示小时的 2 位数字（需要前导零）。目前所有合法规定的时区都不会超过 24 小时制。因此，仅当分钟属性的值为零时，小时属性的值才能为 24。
  - *mm* 用于表示分钟的 2 位数字。如果小时属性的值为 24，那么分钟属性的值必须为零。
  - **+** 指示比 UTC 超前的时区中的指定时刻超前 *hh* 小时 *mm* 分钟。
  - **-** 指示比 UTC 落后的时区中的指定时刻落后 *hh* 小时 *mm* 分钟。
- 文字 Z 表示 UTC 中的时间，Z 表示祖鲁时间，与 UTC 等价。对时区指定 Z 相当于指定 +00:00 或 -00:00。

dayTimeDuration 数据类型

xdt:dayTimeDuration 数据类型表示用天数、小时数、分钟数和秒数部分表示的持续时间。它派生自 xs:duration 数据类型。

此数据类型可表示的范围在 -P83333333333333Y3M11574074074DT1H46M39.999999S 到 P83333333333333Y3M11574074074DT1H46M39.999999S 或 -999999999999999 个月 -999999999999999.999999 秒到 999999999999999 个月 999999999999999.999999 秒之间。

xdt:dayTimeDuration 的词法格式为 P*n*DT*n*H*n*M*n*S，它是 *ISO 8601* 格式的缩减形式。下列缩写用于描述此格式:

P 持续时间指示符。

***n*D** *n* 是无符号整数，用于表示天数。

T 日期和时间分隔符。

***n*H** *n* 是无符号整数，用于表示小时数。

***n*M** *n* 是无符号整数，用于表示分钟数。

***n*S** *n* 是无符号整数，用于表示秒数。如果出现小数点，那么必须后跟 1 到 6 位数字以表示秒的小数部分。

例如，以下格式指示 3 天又 10 小时 30 分钟。

```
P3DT10H30M
```

以下格式指示负 120 天的持续时间:

```
-P120D
```

可选前导减号（-）指示负持续时间。如果省略该符号，那么采用正持续时间。

允许使用此格式的缩减精度和截断表示，但它们必须符合下列要求:

- 如果表达式中的天数、小时数、分钟数或秒数等于零，那么数字及其相应指示符可以省略。但是，必须有至少其中一个数字及其指示符。
- 秒数部分可能有小数部分。
- 当且仅当所有时间项都不存在时，指示符 T 一定不存在。指示符 P 必须始终存在。

例如，允许使用下列格式:

```
P13D
PT47H
P3DT2H
-PT35.89S
P4DT251M
```

不允许使用格式 P-134D，但允许使用格式 -P1347D。

DB2 数据库系统以规范化格式存储 xdt:dayTimeDuration 值。在规范化格式中，秒数和分钟数部分小于 60，小时数部分小于 24。每 60 秒转换为 1 分钟，每 60 分钟转换为 1 小时，而每 24 小时转换为 1 天。例如，以下 XQuery 表达式调用一个构造函数，该构造函数指定 dayTimeDuration 63 天 55 小时 81 秒:

```
  xquery
xdt:dayTimeDuration("P63DT55H81S")
```

在持续时间中，55 小时转换为 2 天又 7 小时，81 秒转换为 1 分 21 秒。表达式返回规范化 dayTimeDuration 值 P65DT7H1M21S。

decimal 数据类型

xs:decimal 数据类型表示可用十进制数字表示的实数子集。它派生自数据类型 xdt:anyAtomicType。

xs:decimal 的词法格式是限定长度的十进制数序列，通过充当小数指示符的句点隔开。允许使用可选前导符号。如果符号被省略，那么表示正号（+）。前导和结尾零都是可选的。如果分数部分为零，那么句点及任何后续零可以省略。下列数字是此数据类型的有效示例:

```
-1.23
12678967.543233
+100000.00
210
```

double 数据类型

xs:double 数据类型的模式是根据 IEEE 双精度 64 位浮点类型确定的。它派生自数据类型 xdt:anyAtomicType。

xs:double 的基本值空间由范围在 -1.7976931348623158e+308 到 -2.2250738585072014e-308 之间以及范围在 +2.2250738585072014e-308 到 +1.7976931348623158e+308 之间的值组成。xs:double 的值空间还包括下列特殊值: 正无穷，负无穷，正零，负零和非数字（NaN）。

xs:double 的词法格式是一个尾数，并且可选择后跟字符 E 或 e 及指数。指数必须是整数。尾数必须是十进制数。指数和尾数的表示必须遵循 xs:integer 和 xs:decimal 的词法规则。如果 E 或 e 及后跟的指数被省略，那么会假定指数值为 0。

零的词法格式可以采用正号或负号。下列文字是此数据类型的有效示例: `-1E4`、`1267.43233E12`、`12.78e-2`、`12`、`-0` 和 `0`。

特殊值正无穷、负无穷和非数字的词法格式分别为 INF、-INF 和 NaN。正无穷的词法格式不能采用正号。

提示: 特殊值 INF、-INF 和 NaN 没有字面值。可使用 xs:double 类型构造函数通过字符串来构造值 INF、-INF 和 NaN。例如: `xs:double("INF")`。

duration 数据类型

xs:duration 数据类型表示通过格列高利历的年、月、日、小时、分钟和秒部分表示的持续时间。它派生自数据类型 xdt:anyAtomicType。

此数据类型可表示的范围在 -P83333333333333Y3M11574074074DT1H46M39.999999S 到 P83333333333333Y3M11574074074DT1H46M39.999999S 或 -999999999999999 个月 -999999999999999.999999 秒到 999999999999999 个月 999999999999999.999999 秒之间。

xs:duration 的词法格式为 *ISO 8601* 扩展格式 P*n*Y*n*M*n*DT*n*H*n*M*n*S。下列缩写用于描述扩展格式:

P 持续时间指示符。

***n*Y** *n* 是无符号整数，用于表示年数。

***n*M** *n* 是无符号整数，用于表示月数。

***n*D** *n* 是无符号整数，用于表示天数。

T 日期和时间分隔符。

***n*H** *n* 是无符号整数，用于表示小时数。

***n*M** *n* 是无符号整数，用于表示分钟数。

***n*S** *n* 是无符号整数，用于表示秒数。如果出现小数点，那么必须后跟 1 到 6 位数字以表示秒的小数部分。

例如，以下格式指示 1 年 2 个月零 3 天又 10 小时 30 分钟的持续时间:

```
P1Y2M3DT10H30M
```

以下格式指示负 120 天的持续时间:

```
-P120D
```

可选前导减号（-）指示负持续时间。如果省略该符号，那么采用正持续时间。

允许使用此格式的缩减精度和截断表示，但它们必须符合下列要求:

- 如果表达式中的年数、月数、天数、小时数、分钟数或秒数等于零，那么数字及其相应指示符可以省略。但是，必须有至少其中一个数字及其指示符。
- 秒数部分可能有小数部分。
- 当且仅当所有时间项都不存在时，指示符 T 一定不存在。
- 指示符 P 必须始终存在。

例如，允许使用下列格式:

```
P1347Y
P1347M
P1Y2MT2H
P0Y1347M
P0Y1347M0D
```

不允许使用格式 P1Y2MT，原因是没有任何时间项。不允许使用格式 P-1347M，但允许使用格式 -P1347M。

DB2 数据库系统以规范化格式存储 xs:duration 值。在规范化格式中，秒数和分钟数部分小于 60，小时数部分小于 24，而月数小于 12。每 60 秒转换为 1 分钟，每 60 分钟转换为 1 小时，每 24 小时转换为 1 天，而每 12 个月转换为 1 年。例如，以下 XQuery 表达式调用一个构造函数，该构造函数指定持续时间 2 个月 63 天 55 小时 91 秒:

```
  xquery
xs:duration("P2M63DT55H91M")
```

在持续时间中，55 小时转换为 2 天又 7 小时，91 分钟转换为 1 小时 31 分钟。表达式返回规范化持续时间值 P2M65DT8H31M。

ENTITY 数据类型

xs:ENTITY 数据类型表示 *XML 1.0 (Third Edition)* 中的 ENTITY 属性类型。它派生自 xs:NCName 数据类型。

xs:ENTITY 的词法格式是不包含冒号的 XML 名称（NCName）。

float 数据类型

xs:float 数据类型的模式是根据 IEEE 单精度 32 位浮点类型确定的。它派生自数据类型 xdt:anyAtomicType。

xs:float 的基本值空间由范围在 -3.4028234663852886e+38 到 -1.1754943508222875e-38 之间以及范围在 +1.1754943508222875e-38 到 +3.4028234663852886e+38 之间的值组成。xs:float 的值空间还包括下列特殊值: 正无穷，负无穷，正零，负零和非数字（NaN）。

xs:float 的词法格式是一个尾数，并且可选择后跟字符 `E` 或 `e` 及指数。指数必须是整数。尾数必须是十进制数。指数和尾数的表示必须遵循 xs:integer 和 xs:decimal 的词法规则。如果 `E` 或 `e` 及后跟的指数被省略，那么会假定指数值为 0。

零的词法格式可以采用正号或负号。下列文字是此数据类型的有效示例: `-1E4`、`1267.43233E12`、`12.78e-2`、`12`、`-0` 和 `0`。

特殊值正无穷、负无穷和非数字的词法格式分别为 INF、-INF 和 NaN。正无穷的词法格式不能采用正号。

提示: 特殊值 INF、-INF 和 NaN 没有字面值。可使用 xs:float 类型构造函数通过字符串来构造值 INF、-INF 和 NaN。例如: `xs:float("INF")`。

gDay 数据类型

xs:gDay 数据类型表示重复出现的格列高利历日期。格列高利历日期是在 *ISO 8601* 中定义的。它派生自数据类型 xdt:anyAtomicType。

此数据类型表示某个月的特定日期。例如，此数据类型可用于指示发薪日是每个月的 15 号。

xs:gDay 的词法格式为 *---ddzzzzzz*，它是 xs:date 的截断表示，未包括月份和年份属性。不允许使用任何前导符号。不允许使用任何其他格式。下列缩写用于描述此格式:

dd 用于表示日期的 2 位数字。

zzzzzz
可选。如果存在，那么表示时区。请参阅第 26 页的『时区指示符』以了解有关此属性的格式的信息。

例如，以下格式指示某个月的 16 号，每个月都会重复出现此日期:

```
---16
```

gMonth 数据类型

xs:gMonth 数据类型表示每年重复出现的格列高利历月份。格列高利历月份是在 *ISO 8601* 中定义的。它派生自数据类型 xdt:anyAtomicType。

此数据类型表示某年的特定月份。例如，此数据类型可用于指示 12 月的圣诞节。

xs:gMonth 的词法格式为 *--mmzzzzzz*，它是 xs:date 的截断表示，未包括年份和日期属性。不允许使用任何前导符号。不允许使用任何其他格式。下列缩写用于描述此格式:

mm 用于表示月份的 2 位数字。

zzzzzz
可选。如果存在，那么表示时区。请参阅第 26 页的『时区指示符』以了解有关此属性的格式的信息。

例如，以下格式指示 12 月，每年都会重复出现的特定月份:

```
--12
```

gMonthDay 数据类型

xs:gMonthDay 数据类型表示重复出现的格列高利历日期。格列高利历日期是在 *ISO 8601* 中定义的。它派生自数据类型 xdt:anyAtomicType。

此数据类型表示某年的特定日期。例如，此数据类型可用于指示每年的 4 月 16 日作为生日。

xs:gMonthDay 的词法格式为 *--mm-ddzzzzzz*，它是 xs:date 的截断表示，未包括年份属性。不允许使用任何前导符号。不允许使用任何其他格式。下列缩写用于描述此格式:

mm 用于表示月份的 2 位数字。

dd 用于表示日期的 2 位数字。

zzzzzz
可选。如果存在，那么表示时区。请参阅第 26 页的『时区指示符』以了解有关此属性的格式的信息。

例如，以下格式指示 4 月 16 日，每年都会重复出现的特定日期:

```
--04-16
```

gYear 数据类型

xs:gYear 数据类型表示格列高利历年份。格列高利历年份是在 *ISO 8601* 中定义的。它派生自数据类型 xdt:anyAtomicType。

xs:gYear 的词法格式为 *yyyyzzzzzz*。此格式是 xs:dateTime 的截断表示，未包括月份、日期或当天时间属性。不允许使用负值日期。下列缩写用于描述此格式:

yyyy
用于表示年份的 4 位数字。有效值是 0001 到 9999 的整数。不允许使用加号（+）。

zzzzzz
可选。如果存在，那么表示时区。请参阅第 26 页的『时区指示符』以了解有关此属性的格式的信息。

例如，以下格式表示格列高利历 2005 年：`2005`。

gYearMonth 数据类型

xs:gYearMonth 数据类型表示特定格列高利历年份的特定格列高利历月份。格列高利历月份是在 *ISO 8601* 中定义的。它派生自数据类型 xdt:anyAtomicType。

xs:gYearMonth 的词法格式为 *yyyy-mmzzzzzz*。此格式是 xs:dateTime 的截断表示，未包括当天时间属性。不允许使用负值日期。下列缩写用于描述此格式：

yyyy
用于表示年份的 4 位数字。有效值是 0001 到 9999 的整数。不允许使用加号（+）。

mm 用于表示月份的 2 位数字。

zzzzzz
可选。如果存在，那么表示时区。请参阅第 26 页的『时区指示符』以了解有关此属性的格式的信息。

例如，以下格式未包括可选时区指示符，用于指示 2005 年的 10 月：

```
2005-10
```

hexBinary 数据类型

xs:hexBinary 数据类型表示十六进制编码的二进制数据。它派生自数据类型 xdt:anyAtomicType。

xs:hexBinary 的词法格式是一个字符序列，其中每个二进制二元都用两位十六进制数字表示。例如，以下格式是 16 位整数 4023 的十六进制编码，其二进制表示为 111110110111：`0FB7`。

ID 数据类型

xs:ID 数据类型表示 *XML 1.0 (Third Edition)* 中的标识属性类型。它派生自 xs:NCName 数据类型。

xs:ID 的词法格式是不包含冒号的 XML 名称（NCName）。

IDREF 数据类型

xs:IDREF 数据类型表示 *XML 1.0 (Third Edition)* 中的 IDREF 属性类型。它派生自 xs:NCName 数据类型。

xs:IDREF 的词法格式是不包含冒号的 XML 名称（NCName）。

int 数据类型

xs:int 数据类型表示小于或等于 2 147 483 647 并且大于或等于 -2 147 483 648 的整数。它派生自 xs:long 数据类型。

xs:int 的词法格式是一个可选符号，后跟限定长度的十进制数序列。如果符号被省略，那么表示正号（+）。下列数字是此数据类型的有效示例：`-1`、`0`、`126789675` 和 `+100000`。

integer 数据类型

xs:integer 数据类型表示小于或等于 9 223 372 036 854 775 807 并大于或等于 -9 223 372 036 854 775 808 的数字。它派生自 xs:decimal 数据类型。

xs:integer 的词法格式是限定长度的十进制数（带有可选前导符号）序列。如果符号被省略，那么表示正号（+）。下列数字是此数据类型的有效示例：`-1`、`0`、`12678967543233` 和 `+100000`。

language 数据类型

xs:language 数据类型表示 *RFC 3066* 定义的自然语言标识。它派生自数据类型 xs:token。

xs:language 的词法格式是用连字符连接的标记字符串。每个标记包含的字符不超过 8 个。第一个标记只能包含字母字符，后续标记可包含字母字符和数字字符。例如，值 en-US 表示美国英语。该字符串符合模式 `[a-zA-Z]{1,8}(-[a-zA-Z0-9]{1,8})*`。

long 数据类型

xs:long 数据类型表示小于或等于 9 223 372 036 854 775 807 并大于或等于 -9 223 372 036 854 775 808 的整数。它派生自数据类型 xs:integer。

xs:long 的词法格式是一个可选符号，后跟限定长度的十进制数序列。如果符号被省略，那么表示正号（+）。下列数字是此数据类型的有效示例：`-1`、`0`、`12678967543233` 和 `+100000`。

Name 数据类型

xs:Name 数据类型表示 XML 名称。它派生自 xs:token 数据类型。

xs:Name 的词法格式是与 *XML 1.0 (Third Edition)* 中的名称产品相匹配的字符串。

NCName 数据类型

xs:NCName 数据类型表示不使用冒号的 XML 名称。它派生自 xs:Name 数据类型。

xs:NCName 的词法格式是不包含冒号的 XML 名称。

negativeInteger 数据类型

xs:negativeInteger 数据类型表示小于零的整数。它派生自数据类型 xs:nonPositiveInteger。

xs:negativeInteger 的词法格式是一个负号（-），后跟限定长度的十进制数序列。此数据类型可表示的范围在 -9223372036854775808 到 -1 之间。下列数字是此数据类型的有效示例：-1、-12678967543233 和 -100000。

NMTOKEN 数据类型

xs:NMTOKEN 数据类型表示 *XML 1.0 (Third Edition)* 中的 NMTOKEN 属性类型。它派生自 xs:token 数据类型。

xs:NMTOKEN 的词法格式是与 *XML 1.0 (Third Edition)* 中的 Nmtoken 产品相匹配的字符串。

nonNegativeInteger 数据类型

xs:nonNegativeInteger 数据类型表示大于或等于零的整数。它派生自 xs:integer 数据类型。

xs:nonNegativeInteger 词法格式是一个可选符号，后跟限定长度的十进制数序列。如果符号被省略，那么表示正号（+）。对于指示零的词法格式，符号可以为正号（+）或负号（-）。在所有其他词法格式中，如果存在符号，那么必须为正号（+）。此数据类型可表示的范围在 0 到 +9223372036854775807 之间。下列数字是此数据类型的有效示例：1、0、12678967543233 和 +100000。

nonPositiveInteger 数据类型

xs:nonPositiveInteger 数据类型表示小于或等于零的整数。它派生自 xs:integer 数据类型。

xs:nonPositiveInteger 的词法格式是一个可选前导符号，后跟限定长度的十进制数序列。对于指示零的词法格式，符号可以为正号（+），也可省略；在所有其他词法格式中，必须呈示负号（-）。此数据类型可表示的范围在 -9223372036854775808 到 0 之间。下列数字是此数据类型的有效示例：-1、0、-12678967543233 和 -100000。

normalizedString 数据类型

xs:normalizedString 数据类型表示空格规范化的字符串。它派生自数据类型 xs:string。

xs:normalizedString 的词法格式是不包含回车符（X'0D'）、换行符（X'0A'）或制表符（X'09'）的字符串。

NOTATION 数据类型

xs:NOTATION 数据类型表示 *XML 1.0 (Third Edition)* 中的 NOTATION 属性类型。它派生自数据类型 xdt:anyAtomicType。

xs:NOTATION 数据类型的词法格式是类型为 xs:QName 的词法格式。

positiveInteger 数据类型

xs:positiveInteger 数据类型表示大于或等于 1 的正整数。它派生自 xs:nonNegativeInteger 数据类型。

xs:positiveInteger 的词法格式是一个可选正号（+），后跟限定长度的十进制数序列。此数据类型可表示的范围在 +1 到 +9223372036854775807 之间。下列数字是此数据类型的有效示例: 1、12678967543233 和 +100000。

QName 数据类型

xs:QName 数据类型表示 XML 限定名 (QName)。QName 包括可选名称空间前缀、用于标识 XML 名称空间的 URI 和本地部分（即 NCName）。它派生自数据类型 xdt:anyAtomicType。

xs:QName 数据类型的词法格式为使用以下格式的字符串: *prefix*:*localName*。下列缩写用于描述此格式:

prefix
可选。名称空间前缀。名称空间前缀必须由名称空间声明绑定至 URI 引用。前缀只能充当名称空间的占位符。如果未指定任何前缀，那么会使用缺省元素/类型名称空间的 URI。

localName
充当限定名称的局部的 NCName。NCName 是没有冒号的 XML 名称。

例如，以下字符串是包含前缀的 QName 的有效词法格式:

```
ns1:emp
```

short 数据类型

xs:short 数据类型表示小于或等于 32767 并大于或等于 -32 768 的整数。它派生自 xs:int 数据类型。

xs:short 的词法格式是一个可选符号，后跟限定长度的十进制数序列。如果符号被省略，那么表示正号（+）。下列数字是此数据类型的有效示例: -1、0、12678 和 +10000。

string 数据类型

xs:string 数据类型表示字符串。它派生自数据类型 xdt:anyAtomicType。

xs:string 的词法格式是一个字符序列，可包含位于 XML 合法字符范围内的任何字符。

time 数据类型

xs:time 数据类型表示每一天重复出现的时刻。它派生自数据类型 xdt:anyAtomicType。

xs:time 的词法格式是 *hh*:*mm*:*ss*.*ssssssszzzzz*。此格式是 xs:dateTime 的截断表示，未包括年份、月份或日期属性。下列缩写用于描述此格式:

hh 用于表示小时的 2 位数字。仅当分钟和秒都表示为零时才允许使用值 24。包括时间 24:00:00 的查询将被视为第二天的 00:00:00 处理。

: 时间部分之间的分隔符。

mm 用于表示分钟的 2 位数字。

ss 用于表示秒的整数部分的 2 位数字。

.ssssss
可选。如果存在，那么为表示秒的小数部分的 1 到 6 位数字。

zzzzzz
可选。如果存在，那么表示时区。请参阅第 26 页的『时区指示符』以了解有关此属性的格式的信息。

例如，以下格式包括可选时区指示符，表示东部标准时间下午 1:20，它比全球标准时间（UTC）提前 5 个小时:

```
13:20:00-05:00
```

token 数据类型

xs:token 数据类型表示标记化字符串。它派生自 xs:normalizedString 数据类型。

xs:token 的词法格式是不包含下列任何字符的字符串:

- 回车符（X'0D'）
- 换行符（X'0A'）
- 制表符（X'09'）
- 前导或结尾空格（X'20'）
- 两个或更多空格的内部序列

unsignedByte 数据类型

xs:unsignedByte 数据类型表示小于或等于 255 的无符号整数。它派生自 xs:unsignedShort 数据类型。

xs:unsignedByte 的词法格式是限定长度的十进制数序列。下列数字是此数据类型的有效示例: 0、126 和 100。

unsignedInt 数据类型

xs:unsignedInt 数据类型表示小于或等于 4 294 967 295 的无符号整数。它派生自 xs:unsignedLong 数据类型。

xs:unsignedInt 的词法格式是限定长度的十进制数序列。下列数字是此数据类型的有效示例: `0`、`1267896754` 和 `100000`。

unsignedLong 数据类型

xs:unsignedLong 数据类型表示小于或等于 9 223 372 036 854 775 807 的无符号整数。它派生自 xs:nonNegativeInteger 数据类型。

xs:unsignedLong 的词法格式是限定长度的十进制数序列。下列数字是此数据类型的有效示例: `0`、`12678967543233` 和 `100000`。

unsignedShort 数据类型

xs:unsignedShort 数据类型表示小于或等于 65 535 的无符号整数。它派生自 xs:unsignedInt 数据类型。

xs:unsignedShort 的词法格式是限定长度的十进制数序列。下列数字是此数据类型的有效示例: `0`、`12678` 和 `10000`。

untyped 数据类型

xdt:untyped 数据类型指示未经过 XML 模式验证的节点。它派生自数据类型 xs:anyType。

如果元素节点注释为 xdt:untyped，那么它的所有后代元素节点也会注释为 xdt:untyped。

untypedAtomic 数据类型

xdt:untypedAtomic 数据类型指示未经过 XML 模式验证的原子值。它派生自数据类型 xdt:anyAtomicType。

数据类型 xdt:untypedAtomic 使用无约束的词法格式。

yearMonthDuration 数据类型

xdt:yearMonthDuration 数据类型表示通过格列高利历的年和月部分表示的持续时间。它派生自 xs:duration 数据类型。

此数据类型可表示的范围在 -P83333333333333Y3M 到 P83333333333333Y3M 或 -999999999999999 个月到 999999999999999 个月之间。

xdt:yearMonthDuration 的词法格式为 `PnYnM`，这是 *ISO 8601* 格式的缩减形式。下列缩写用于描述此格式:

***n*Y** *n* 是无符号整数，用于表示年数。

***n*M** *n* 是无符号整数，用于表示月数。

可选前导减号（-）指示负持续时间。如果省略该符号，那么采用正持续时间。

例如，以下格式指示持续时间 1 年零 2 个月:

```
P1Y2M
```

以下格式指示负 13 个月的持续时间:

```
-P13M
```

允许使用此格式的缩减精度和截断表示，但它们必须符合下列要求:

- 指示符 `P` 必须始终存在。
- 如果任何表达式中的年数或月数等于零，那么数字及其相应指示符可以省略。但是，必须有至少其中一个数字及其指示符（`Y` 或 `M`）。

例如，允许使用下列格式:

```
P1347Y
P1347M
```

不允许使用格式 P-1347M，但允许使用格式 -P1347M。不允许使用 P24YM 和 PY43M，原因是 Y 必须至少有一个前导数字，并且 M 也必须至少有一个前导数字。

DB2 数据库系统以规范化格式存储 xdt:yearMonthDuration 值。在规范化格式中，月数部分小于 12。每 12 个月转换为 1 年。例如，以下 XQuery 表达式调用一个构造函数，该构造函数指定 yearMonthDuration 20 年 30 个月:

```
  xquery
xdt:yearMonthDuration("P20Y30M")
```

在此持续时间中，30 个月转换为 2 年零 6 个月。表达式返回规范化 yearMonthDuration 值 P22Y6M。

第 3 章 序言

序言是一系列声明，用于定义查询的处理环境。序言中的每个声明都会后跟分号（;）。序言是查询的可选部分；有效查询可由不带序言的查询主体组成。

序言包括可选版本声明、名称空间声明和 *setter*，setter 是设置影响查询处理的属性值的可选声明。

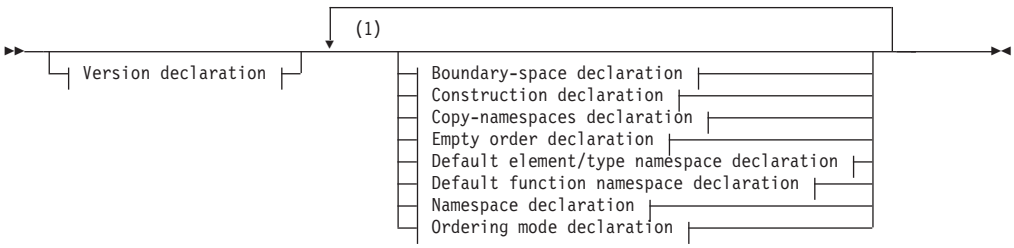
DB2 XQuery 支持边界空间声明，该声明可用于更改查询的处理方式。序言还包括名称空间声明和缺省名称空间声明。

DB2 XQuery 还支持下列 setter。但是，因为 DB2 XQuery 在每种情况下仅支持一个选项，所以它们不会更改处理环境:

- 构造声明
- 复制名称空间声明
- 空排序声明
- 排序方式声明

如果版本声明存在，那么必须出现在序言开头。setter 及其他声明可按任意顺序出现在序言中版本声明之后的位置。

语法



注:

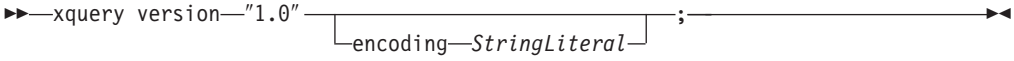
1 每个声明只能指定一次，但名称空间声明除外。

版本声明

版本声明出现在查询开头，用于标识处理查询所需的 XQuery 语法和语义的版本。版本声明可包含编码声明，但编码声明会被 DB2 XQuery 忽略。

如果存在，那么版本声明必须在序言开头。DB2 XQuery 唯一支持的版本为“1.0”。

语法



1.0
指定处理查询需要版本 1.0 的 XQuery 语法和语义。

StringLiteral
指定表示编码名称的字符串文字。因为 *StringLiteral* 的值被忽略，所以指定编码声明不会影响查询。DB2 XQuery 总是假定编码为 UTF-8。

示例

以下版本声明指示查询必须由支持 XQuery 版本 1.0 的实现来处理:

```
xquery version "1.0";
```

边界空格声明

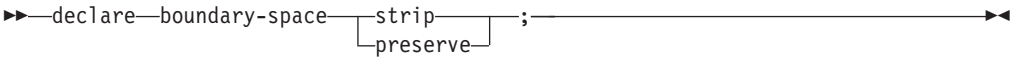
查询序言中的边界空格声明会对查询设置边界空格策略。*边界空格策略*会控制元素构造函数对空格的处理方式。

*边界空格*包括元素构造函数中的标记或带括号表达式之间的边界中自己出现的所有空格字符。

边界空格策略可指定构造元素时是保留边界空格还是除去边界空格。如果未指定边界空格声明，那么缺省行为是在构造元素时除去边界空格。

对于每个查询，序言只能包含一个边界空格声明。

语法



strip
指定构造元素时除去边界空格。

preserve
指定构造元素时保留边界空格。

示例

以下边界空格声明指定构造元素时将保留边界空格:

```
declare boundary-space preserve;
```

构造声明

查询序言中的构造声明将为查询设置构造方式。*构造方式*控制如何对复制的元素和属性节点指定类型注释，以形成新的构造节点的内容。

在 DB2 XQuery 中，已构造元素节点的构造方式始终为 **strip**。对于 DB2 XQuery，当构造方式为 **strip** 时，已构造元素节点的类型为 xdt:untypedAtomic；在构造节点期间复制的所有元素节点都将接收到 xdt:untypedAtomic 类型，而在构造节点期间复制的所有属性节点都将接收到 xdt:untypedAtomic 类型。

指定除 **strip** 之外的值的构造声明会产生错误。对于每个查询，序言只能包含一个构造声明。

语法

strip
对于 DB2 XQuery，指定已构造元素节点的类型为 xdt:untypedAtomic；在构造节点期间复制的所有元素节点都将接收到 xdt:untypedAtomic 类型，而在构造节点期间复制的所有属性节点都将接收到 xdt:untypedAtomic 类型。

示例

以下构造声明有效，但不会更改元素构造的缺省行为:

```
declare construction strip;
```

复制名称空间声明

复制名称空间方式控制元素构造函数复制现有元素节点时指定的名称空间绑定。

在 DB2 XQuery 中，复制名称空间方式始终为 **preserve** 和 **inherit**。设置 **preserve** 用于指定原始元素的所有名称空间作用域将保留在新副本中。缺省名称空间的处理方式与任何其他名称空间绑定相同：复制的节点保留其缺省名称空间或缺少缺省名称空间。设置 **inherit** 指定复制的节点从构造节点继承名称空间作用域。如果发生冲突，那么从原始节点保留的名称空间绑定优先。

指定 **preserve** 和 **inherit** 以外的值的复制名称空间声明将产生错误。对于每个查询，序言只能包含一个复制名称空间声明。

语法

preserve
指定原始元素的名称空间作用域将保留在新副本中。

inherit
指定复制的节点将继承构造节点的名称空间作用域。

示例

以下复制名称空间声明有效，但不会更改元素构造的缺省行为:

```
declare copy-namespaces preserve, inherit;
```

缺省元素/类型名称空间声明

查询序言中的缺省元素/类型名称空间声明将指定名称空间，该名称空间用于元素和类型名称的无前缀 QName（限定名称）。

查询序言只能包含一个缺省元素/类型名称空间声明。除非此声明被直接元素构造函数中的名称空间声明属性覆盖，否则声明在声明查询的作用域内。如果未声明任何缺省元素/类型名称空间，那么无前缀元素和类型名称不在任何名称空间内。

缺省元素/类型名称空间不适用于非限定属性名称。无前缀属性名称和变量名称不在任何名称空间内。

语法

element
指定声明为缺省元素/类型名称空间声明。

URILiteral
指定表示名称空间的 URI 的字符串文字。字符串文字必须是有效 URI 或零长度字符串。如果缺省元素/类型名称空间声明中的字符串文字是零长度字符串，那么无前缀元素和类型名称不在任何名称空间内。

示例

以下声明指定元素和类型名称的缺省名称空间是与 URI `http://posample.org` 相关联的名称空间:

```
declare default element namespace "http://posample.org";
<name>Snow boots</name>
```

执行示例中的查询时，新创建的节点（元素节点 `name`）在与名称空间 URI `http://posample.org` 相关联的名称空间中。

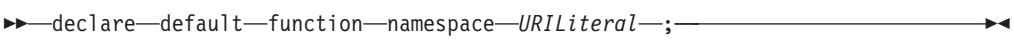
缺省函数名称空间声明

查询序言中的缺省函数名称空间声明指定用于函数调用中无前缀函数名的名称空间 URI。

查询序言只能包含一个缺省函数名称空间声明。如果未声明任何缺省函数名称空间，那么缺省函数名称空间是 XPath 和 XQuery 函数的名称空间 `http://www.w3.org/2005/xpath-functions`。如果声明了缺省函数名称空间，那么可在不指定前缀的情况下在缺省函数中调用任何函数。

如果无前缀函数调用的局部名与缺省函数名称空间中的函数不匹配，那么 DB2 XQuery 会产生错误。

语法



function
指定该声明是缺省函数名称空间声明。

URILiteral
指定表示名称空间的 URI 的字符串文字。字符串文字必须是有效 URI 或零长度字

符串。如果缺省函数名称空间声明中的字符串文字为零长度字符串，那么所有函数调用都必须使用加前缀函数名，原因是每个函数都在同一名称空间中。

示例

以下声明指定缺省函数名称空间与 URI `http://www.ibm.com/xmlns/prod/db2/functions` 相关联:

```
declare default function namespace "http://www.ibm.com/xmlns/prod/db2/functions";
```

在此示例的查询主体中，不用在函数名中加入前缀就可以引用缺省函数名称空间中的任何函数。此缺省函数名称空间包括函数 `xmlcolumn`，所以可输入 `xmlcolumn('T1.MYDOC')` 来代替 `db2-fn:xmlcolumn('T1.MYDOC')`。但是，因为此示例中的缺省函数名称空间不再与 XQuery 函数的名称空间相关联，所以调用 XQuery 内置函数时需要指定前缀。例如，必须输入 `fn:current-date()` 来代替 `current-date()`。

空排序声明

在处理 FLWOR 表达式的 **order by** 子句时，查询序言中的空排序声明用于控制空序列或 NaN 值是解释为最高值还是最低值。

在 DB2 XQuery 中，处理 FLWOR 表达式中的 **order by** 子句时空序列总是解释为最高值。NaN 值解释为大于空序列以外的所有其他值。此设置不会被覆盖。指定 **empty greatest** 以外的值的空排序声明会产生错误。对于每个查询，查询序言只能包含一个空排序声明。

语法

greatest
处理 FLWOR 表达式中的 **order by** 子句时空序列总是解释为最高值。NaN 值解释为大于空序列以外的所有其他值。

示例

以下空排序声明有效:

```
declare default order empty greatest;
```

排序方式声明

查询序言中的*排序方式声明*设置查询的排序方式。*排序方式*会定义查询结果中的节点排序。

因为 DB2 XQuery 不支持 *XQuery : An XML Query Language* 中定义的排序方式，所以如果排序方式声明存在，那么必须指定无序。对于 DB2 XQuery 中用于控制查询结果顺序的规则，请参阅第 48 页的『XQuery 表达式中结果的顺序』。

查询序言只能包含一个排序方式声明。指定无序结果以外的值的排序方式声明会产生错误。

语法

unordered
指定 *XQuery 1.0: An XML Query Language* 中的排序方式规则不起作用。对于 DB2 XQuery 中用于控制查询结果顺序的规则，请参阅第 48 页的『XQuery 表达式中结果的顺序』。

示例

以下声明有效，但不会更改排序的缺省行为，原因是 DB2 XQuery 仅支持无序方式:

```
declare ordering unordered;
```

名称空间声明

查询序言中的名称空间声明会声明名称空间前缀并使该前缀与名称空间 URI 相关联。

前缀与名称空间 URI 之间的关联称为*名称空间绑定*。在名称空间声明中绑定的名称空间将添加至静态已知名称空间。*静态已知名称空间*包含处理查询期间可用于解析名称空间前缀的所有名称空间绑定。

除非名称空间声明被直接元素构造函数中的名称空间声明属性覆盖，否则此声明在声明查询的作用域内。如果查询序言中存在同一名称空间前缀的多个声明，那么会产生错误。

语法

prefix
指定绑定至 *URILiteral* 指定的 URI 的名称空间前缀。该名称空间前缀在限定名称（QName）中用于标识元素、属性、数据类型或函数的名称空间。

前缀 `xmlns` 和 `xml` 已被系统保留，不能在名称空间声明中指定为前缀。

URILiteral
指定前缀绑定至的 URI。*URILiteral* 必须为包含有效 URI 的非零长度文字串。

示例

以下查询包括一个名称空间声明，用于声明名称空间前缀 `ns1` 并使其与名称空间 URI `http://posample.org` 相关联:

```
declare namespace ns1 = "http://posample.org";
<ns1:name>Thermal gloves</ns1:name>
```

执行示例中的查询时，新创建的节点（元素节点 `name`）在与名称空间 URI `http://posample.org` 相关联的名称空间中。

预先声明的名称空间前缀

XQuery 有一些预先声明的名称空间前缀，处理每个查询前它们在静态已知名称空间中。可在没有显式声明的情况下使用任何预先声明的名称空间前缀。DB2 XQuery 的预先声明的名称空间前缀包括下表中显示的前缀和 URI 对:

表 11. DB2 XQuery 中的预先声明名称空间

前缀	URI	描述
xml	`http://www.w3.org/XML/1998/namespace`	XML 保留名称空间
xs	`http://www.w3.org/2001/XMLSchema`	XML 模式名称空间
xsi	`http://www.w3.org/2001/XMLSchema-instance`	XML 模式实例名称空间
fn	`http://www.w3.org/2005/xpath-functions`	缺省函数名称空间
xdt	`http://www.w3.org/2005/xpath-datatypes`	XQuery 类型名称空间
db2-fn	`http://www.ibm.com/xmlns/prod/db2/functions`	DB2 函数名称空间

可通过在查询序言中指定名称空间声明来覆盖预先声明的名称空间前缀。但是，不能覆盖与前缀 `xml` 相关联的 URI。

第 4 章 表达式

表达式是查询的基本构建块。表达式可单独使用，也可与其他表达式组合以构成复杂查询。DB2 XQuery 支持使用多种表达式来处理 XML 数据。

表达式求值和处理

表达式的处理中通常包含若干操作。这些操作包括从节点抽取原子值，使用类型提升和子类型替换以获取期望类型的值，以及计算序列的布尔值。

在 DB2 XQuery 中，只能在变换表达式的 **modify** 子句中使用更新表达式。有关 XQuery 变换表达式及更新表达式处理的信息，请参阅第 104 页的『变换表达式』和第 101 页的『在变换表达式中使用更新操作』。

动态上下文和焦点

表达式的动态上下文是对表达式求值时可用的信息。焦点由上下文项、上下文位置和上下文大小组成，是动态上下文的重要组成部分。

DB2 XQuery 处理序列中的每一项时，焦点会随之改变。焦点包含下列信息:

上下文项
当前处理的原子值或节点。可通过由单个点组成的上下文项表达式（.）来检索上下文项。

上下文位置
序列中当前处理的上下文项的位置。可通过 fn:position() 函数检索上下文项。

上下文大小
序列中当前处理的项数。可通过 fn:last() 函数检索上下文大小。

优先顺序

XQuery 语法定义运算符与表达式之间的内置优先顺序。如果优先顺序较低的表达式用作优先顺序较高的表达式的操作数，那么优先顺序较低的表达式必须用圆括号括起来。

下表按从低到高的优先顺序列示 XQuery 运算符和表达式。关联性列指示具有相等优先顺序的运算符或表达式的应用顺序。

表 12. DB2 XQuery 中的优先顺序

运算符或表达式	关联性
,（逗号）	从左至右
:=（赋值）	从右至左
FLWOR, some, every, if	从左至右
or	从左至右
and	从左至右
eq, ne, lt, le, gt, ge, =, !=, <, <=, >, >=, is, <<, >>	从左至右
to	从左至右

表 12. DB2 XQuery 中的优先顺序 (续)

运算符或表达式	关联性
+, -	从左至右
*, div, idiv, mod	从左至右
union, |	从左至右
intersect, except	从左至右
可转型	从左至右
强制类型转换	从左至右
-(unary), +(unary)	从右至左
?	从左至右
/, //	从左至右
[ ], ( ), {}	从左至右

XQuery 表达式中结果的顺序

使用 DB2 XQuery 时，某些种类的 XQuery 表达式会按确定顺序返回序列，而其他种类的表达式会按不确定的顺序返回序列。

下列种类的表达式按确定顺序返回序列:

- 包含显式 **order by** 子句的 FLWOR 表达式按指定顺序返回结果。例如，以下表达式按价格升序返回产品元素序列:

  ```
  for $p in /product
  order by $p/price
  return $p
  ```

- 添加元素并使用显式位置关键字的更新表达式返回结果时，添加的元素会放在指定位置。例如，以下更新表达式使用 **as first into** 关键字，并插入元素 `<status>current</status>` 作为 customerinfo 元素中的第一个项元素:

  ```
  xquery
  transform
  copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1001')
  modify
    do insert <status>current</status> as first into $mycust/customerinfo
  return $mycust
  ```

- 将序列与 **union**、**intersect** 或 **except** 运算符组合到一起的表达式会按文档顺序返回结果。
- 符合下列条件的路径表达式会按文档顺序返回结果:
  - 路径表达式仅包含正向轴步骤。
  - 路径表达式起源于单个节点，例如，可通过函数调用或变量引用生成。
  - 路径表达式中的任何步骤都只包含单个谓词。
  - 路径表达式不包含 fn:position 函数调用或 fn:last 函数调用。

  以下示例是按文档顺序返回结果的路径表达式，假定变量 *$bib* 绑定至单个元素。

  ```
  $bib/book[title eq "War and Peace"]/chapter
  ```

- 范围表达式是包含 **to** 运算符的表达式，并且会按升序返回整数序列。例如，`15 to 25`。
- 对于包含逗号运算符的表达式，如果操作数都是具有确定顺序的序列，那么会按操作数顺序返回结果。例如，以下表达式返回序列 (5, 10, 15, 16, 17, 18, 19, 20, 25):

```
(5, 10, 15 to 20, 25)
```

- 对于包含按确定顺序返回结果的操作数表达式的其他表达式，会按确定顺序返回结果。例如，假定变量 *$pub* 绑定至单个元素，那么以下条件表达式会返回有序结果，原因是 then 和 else 子句中的路径表达式返回有序结果:

```
if ($pub/type eq "journal")
  then $pub/editor
  else $pub/author
```

如果在以上列表中未列示的表达式返回多项，那么序列中的项数是不确定的。

表 13. XQuery 表达式中结果的顺序总结

表达式种类	具有确定排序的条件	结果顺序	示例
FLWOR	显式 **order by** 子句	由 **order by** 子句确定	以下表达式按价格升序排列返回产品元素序列: `for $p in /product order by $p/price return $p`
更新表达式	在添加元素时使用指定位置的关键字	由更新表达式关键字确定	使用关键字 **as last into** 插入元素时，该元素将添加为指定节点的最后一个子代。
使用 **union**、**intersect** 或 **except** 运算符的表达式	无	文档顺序	`$managers union $students`
路径表达式	• 路径表达式仅包含正向轴步骤。 • 路径表达式起源于单个节点，例如，可通过函数调用或变量引用生成。 • 路径表达式中的任何步骤都只包含单个谓词。 • 路径表达式不包含 fn:position 函数调用或 fn:last 函数调用。	文档顺序	以下示例是按文档顺序返回结果的路径表达式，假定变量 *$bib* 绑定至单个元素。 `$bib/book [title eq "War and Peace"] /chapter`
范围表达式，即包含 **to** 运算符的表达式	无	按升序排列的整数序列	`15 to 25`
包含逗号运算符的表达式	所有操作数是带有确定顺序的序列	返回按操作数顺序排列的结果	`(5, 10, 15 to 20, 25)`

表 13. XQuery 表达式中结果的顺序总结 (续)

表达式种类	具有确定排序的条件	结果顺序	示例
其他表达式	操作数表达式全部返回按确定顺序排列的结果	由嵌套表达式的结果顺序确定	假定变量 *$pub* 绑定至单个元素，那么以下条件表达式会返回有序结果，原因是 then 和 else 子句中的路径表达式返回有序结果: `if ($pub/type eq "journal") then $pub/editor else $pub/author`

注: 如果对没有确定顺序的序列应用位置谓词，那么结果是不确定的，这意味着可能选择序列中的任意项。

原子化

*原子化*是将项序列转换为原子值序列的过程。每当需要原子值序列时，表达式就会使用原子化。

通过应用下列规则，序列中的每一项都会转换为原子值:

- 如果该项是原子值，那么返回原子值。
- 如果该项是节点，那么返回类型值。节点的*类型值*是可从该节点抽取的零个或零个以上原子值的序列。如果节点没有类型值，那么会返回错误。

序列的隐式原子化与对序列显式调用 fn:data 函数的结果是相同的。

例如，以下序列包含节点与原子值的组合:

```
("Some text",<anElement xsi:type="string">More text</anElement>,
<anotherElement xsi:type="decimal">1.23</anotherElement>,1001)
```

对此序列应用原子化会生成下列原子值序列:

```
("Some text", "More text", 1.23, 1001)
```

下列 XQuery 表达式使用原子化将各项转换为原子值:

- 算术表达式
- 比较表达式
- 使用预期类型为原子的自变量的函数调用
- 强制类型转换表达式
- 用于各种节点的构造函数表达式
- FLWOR 表达式中的 **order by** 子句
- 类型构造函数

子类型替换

*子类型替换*是指使用其动态类型派生自期望类型的值。

子类型替换不会更改值的实际类型。例如，如果所使用值的类型为 xs:integer 而该值的预期类型为 xs:decimal，那么该值会将保留其 xs:integer 类型。

在以下示例中，fn:compare 函数会将 xs:string 值与 xs:NCName 值进行比较:

```
fn:compare("product", xs:NCName("product"))
```

返回的值为 0，这意味着自变量的比较结果为相等。尽管 fn:compare 函数期望类型为 xs:string 的自变量，但可使用类型为 xs:NCNAME 的值调用该函数，原因是此类型派生自 xs:string。

每次对表达式传递派生自期望类型的值时，会使用子类型替换。

类型提升

*类型提升*是将原子值从其原始类型转换为表达式期望类型的过程。XQuery 会在对接受数字或字符串操作数的函数调用、**order by** 子句和运算符求值期间使用类型提升。

XQuery 允许执行下列类型提升:

数字类型提升:

类型为 xs:float（或根据 xs:float 中的限制派生的任何类型）的值可提升至类型 xs:double。结果是与原始值相同的 xs:double 值。

类型为 xs:decimal（或根据 xs:decimal 中的限制派生的任何类型）的值可提升至类型 xs:float 或 xs:double。此提升的结果是通过将原始值转换为必需类型实现的。此类提升可能会导致精度下降。

在以下示例中，包含 xs:double 值 13.54e-2 和 xs:decimal 值 100 的序列将传递至 fn:sum 函数，该函数会返回类型为 xs:double 的值:

```
fn:sum(xs:double(13.54e-2), xs:decimal(100))
```

URI 类型提升:

类型为 xs:anyURI（或根据 xs:anyURI 中的限制派生的任何类型）的值可提升至类型 xs:string。此提升的结果是通过将原始值转换为类型 xs:string 实现的。

在以下示例中，URI 值将提升至期望类型 xs:string，而函数会返回 18:

```
fn:string-length(xs:anyURI("http://example.com"))
```

注意，使用下列方式时，类型提升和子类型替换会有所不同:

- 对于类型提升，实际上会将原子值从其原始类型转换为表达式期望的类型。
- 对于子类型替换，可使用派生自该类型的值调用期望特定类型的表达式。但是，该值仍保留其原始值。

有效布尔值

序列的*有效布尔值（EBV）*是在处理需要使用布尔值的表达式期间隐式计算的。值的 EBV 是通过对值应用 fn:boolean 函数确定的。

下表描述对特定值类型返回的 EBV。

表 14. 对 XQuery 中的特定值类型返回的 EBV

值的描述	返回的 EBV
空序列	false

表 14. 对 XQuery 中的特定值类型返回的 EBV (续)

值的描述	返回的 EBV
第一项为节点的序列	true
类型为 xs:boolean 或派生自 xs:boolean 的单个值	false – 如果 xs:boolean 值为 false
	true – 如果 xs:boolean 值为 true
类型为 xs:string 或 xdt:untypedAtomic 或派生自其中一种类型的单个值	false – 如果值长度为零
	true – 如果值长度大于零
任意数字类型或派生自数字类型的单个值	false – 如果值为 NaN 或者其数值等于零
	true – 如果该值的数值不等于零
所有其他值	错误
注: 对于包含至少一个节点和至少一个原子值的序列，其有效布尔值在顺序不可预测的查询中是不确定的。	

序列的有效布尔值是在处理下列表达式类型时隐式计算的:

- 逻辑表达式（**and** 和 **or**）
- fn:not 函数
- FLWOR 表达式的 **where** 子句
- 特定类型的谓词，如 `a[b]`
- 条件表达式（**if**）
- 定量表达式（**some** 和 **every**）

主表达式

主表达式是语言的基本主体。它们包括文字、变量引用、带括号表达式、上下文项表达式、构造函数和函数调用。

文字

文字是原子值的直接语法表示。DB2 XQuery 支持两种类型的文字: 数字文字和字符串文字。

*数字文字*是类型为 xs:integer、xs:decimal 或 xs:double 的原子值:

- 不包含小数点（.）及 e 或 E 字符的数字文字是类型为 xs:integer 的原子值。例如, `12` 是数字文字。
- 包含小数点（.）但不包含 e 或 E 字符的数字文字是类型为 xs:decimal 的原子值。例如, `12.5` 是数字文字。
- 包含 e 或 E 字符的数字文字是类型为 xs:double 的原子值。例如, `125E2` 是数字文字。

数字文字的值是根据 XML 模式规则解释的。

*字符串文字*是类型为 xs:string 的原子值，括在定界单引号（'）或双引号（"）中。字符串文字可包括预定义实体引用和字符引用。例如，下列字符串文字都是有效字符串文字:

```
"12.5"
"He said, ""Let it be."""
'She said: "Why should I?"'
"Ben &amp;amp; Jerry&amp;apos;s"
"&amp;#8364;65.50" (: denotes the string €65.50 :)
```

提示: 要在用单引号定界的字符串文字中加入单引号，请指定两个相邻单引号。同样，要在用双引号定界的字符串文字中加入双引号，请指定两个相邻双引号。

在字符串文字中，行结尾是根据 *XML 1.0 (Third Edition)* 的规则规范化的。包含回车符（X'0D'）并后跟换行符（X'0A'）的任何两字符序列都将转换为单个换行符（X'0A'）。未后跟换行符（X'0A'）的任何回车符（X'0D'）将转换为单个换行符（X'0A'）。

如果要具体化的值没有文字表示，那么可使用构造函数或内置函数返回该值。下列函数和构造函数会返回没有文字表示的值:

- 使用内置函数 fn:true() 和 fn:false() 会分别返回布尔值 `true` 和 `false`。还可使用构造函数 xs:boolean(″false″) 和 xs:boolean(″true″) 返回这些值。
- 使用构造函数 `xs:date("2005-04-16")` 会返回类型为 xs:date 并且值表示 2005 年 4 月 16 日的项。
- 使用构造函数 `xdt:dayTimeDuration("PT4H")` 会返回类型为 xdt:dayTimeDuration 并且值表示四个小时持续时间的项。
- 使用构造函数 `xs:float("NaN")` 会返回特殊浮点值“Not a Number”。
- 使用构造函数 `xs:double("INF")` 会返回特殊双精度值“positive infinity”。

预定义实体引用

*预定义实体引用*是一个简短字符序列，表示在 DB2 XQuery 中具有一定语法意义的字符。

预定义实体引用以和号（&）开头，以分号（;）结尾。处理字符串文字时，每个预定义实体引用将替换为它所表示的字符。

下表列示 DB2 XQuery 识别的预定义实体引用。

表 15. DB2 XQuery 中的预定义实体引用

实体引用	表示的字符
&lt;	<
&gt;	>
&amp;	&
&quot;	″
&apos;	’

字符引用

*字符引用*是对 Unicode 字符的 XML 样式引用，Unicode 字符由其十进制或十六进制代码点标识。

字符引用以 &#x 或 &# 开头，以分号（;）结尾。如果字符引用以 &#x 开头，那么终止分号（;）之前的数字和字母将根据 *ISO/IEC 10646* 标准表示为字符代码点的十六进制形式。如果字符引用以 &# 开头，那么终止分号（;）之前的数字将表示为字符代码点的十进制形式。

示例

字符引用 € 或 € 表示欧元符号（€）。

变量引用

变量引用是将美元符号（$）用作前缀的 NCName。对查询求值时，每个变量引用会解析为绑定至该变量的值。每个变量引用必须与变量作用域中引用位置处的名称相匹配。

变量是使用下列方式添加至变量作用域的:

- 可通过主语言环境、SQL/XML、XMLQUERY 函数、XMLTABLE 函数或 XMLEXISTS 谓词向变量作用域添加变量。对于由 SQL/XML 添加的变量，除非该变量被 XQuery 表达式中同一变量的另一绑定覆盖，否则该变量在整个查询的作用域中。
- 变量可根据 XQuery 表达式绑定至某个值。可绑定变量的表达式种类包括 FLWOR 表达式和定量表达式。在执行函数体之前，函数调用也会将值绑定至函数的形参。由 XQuery 表达式绑定的变量在该表达式的作用域中。

  变量名称只能在 FLWOR 表达式中声明一次。例如，DB2 XQuery 不支持以下表达式:

  ```
  for $i in (1, 2)
  for $i in ("a", "b")
  return $i
  ```

如果变量引用与作用域中的两个或更多变量绑定相匹配，那么该引用会引用内部绑定，即作用域更小的绑定。

提示: 为提高代码的可读性，请对查询中的变量使用唯一名称。

示例

在以下示例中，FLWOR 表达式将变量 $seq 绑定至序列 (10, 20, 30):

```
let $seq := (10, 20, 30)
return $seq[2];
```

返回的值为 20。

带括号表达式

圆括号可用于在包含多个运算符的表达式中强制实现特定求值顺序。

例如，表达式 (2 + 4) * 5 的求值结果为 30，原因是先对带括号表达式 (2 + 4) 求值，然后对其结果乘 5。如果没有圆括号，那么表达式 2 + 4 * 5 的求值结果为 22，原因是乘法运算符的优先顺序高于加法运算符。

空圆括号指示空序列。

上下文项表达式

上下文项表达式由单个句点字符（.）组成。上下文项表达式会求值为当前正在处理的项，又称为*上下文项*。

上下文项可以是节点或原子值。只能在路径表达式和谓词表达式中定义上下文项。

示例

以下示例包含上下文项表达式，它将对范围表达式 `1 to 100` 所返回序列中的每项调用模数运算符:

```
(1 to 100)[. mod 5 eq 0]
```

此示例将生成 1 至 100 之间可被 5 整除的整数序列。

函数调用

函数调用由 QName 组成，后跟用圆括号括起来的零个或零个以上表达式（称为自变量）。DB2 XQuery 支持调用内置 XQuery 函数和 DB2 内置函数。

内置 XQuery 函数在绑定至前缀 `fn` 的名称空间 `http://www.w3.org/2005/xpath-functions` 中。DB2 内置函数在绑定至前缀 `db2-fn` 的名称空间 `http://www.ibm.com/xmlns/prod/db2/functions` 中。如果函数调用中的 QName 没有名称空间前缀，那么该函数必定在缺省函数名称空间中。除非名称空间被查询序言中的缺省函数声明覆盖，否则缺省函数名称空间是内置 XQuery 函数的名称空间（绑定至前缀 `fn`）。

要点: 因为函数调用的自变量是用逗号分隔的，所以必须使用圆括号将包含顶级逗号运算符的自变量表达式括起来。

下列步骤说明 DB2 XQuery 对函数求值时所用的过程:

1. DB2 XQuery 对函数调用中作为自变量传递的每个表达式求值，并返回每个表达式的值。
2. 对每个自变量返回的值将转换为该自变量应使用的数据类型。如果自变量应该为原子值或原子值序列，那么 DB2 XQuery 会使用下列规则将自变量的值转换为预期类型:
   a. 将对给定值进行原子化。这会产生原子值序列。
   b. 原子序列中类型为 xdt:untypedAtomic 的每项的数据类型将转换为预期原子类型。对于应该为原子自变量的内置函数，类型为 xdt:untypedAtomic 的自变量的数据类型将转换为 xs:double。
   c. 将对原子序列中任何可提升至预期原子类型的数字项应用数字类型提升。数字项包括类型为 xs:integer（或派生自 xs:integer）、xs:decimal、xs:float 或 xs:double 的项。
   d. 如果预期类型为 xs:string，那么原子序列中类型为 xs:anyURI（或派生自 xs:anyURI）的每项将提升至 xs:string。
3. 将使用函数自变量的转换值对函数求值。函数调用将生成函数声明的返回类型的实例，或者产生错误。

示例

使用字符串自变量的函数调用: 以下函数调用采用类型为 xs:string 的自变量，并返回类型为 xs:string 的值，其中所有字符为大写:

```
fn:upper-case($ns1_customerinfo/ns1:addr/@country)
```

在此示例中，传递至 fn:upper-case 函数的自变量为路径表达式。调用该函数时，会对路径表达式求值，并且会对生成的节点序列进行原子化。序列中的每个原子值将转换为预期类型 xs:string。将对该函数求值，并返回类型为 xs:string 的原子值序列。

使用序列自变量的函数调用: 以下函数采用序列 (1, 2, 3) 作为单个自变量。

```
fn:max((1, 2, 3))
```

因为函数 fn:max 需要原子值序列形式的单个自变量，所以需要嵌套圆括号。返回的值为 3。

路径表达式

*路径表达式*标识 XML 树中的节点。DB2 XQuery 中的路径表达式以 XPath 2.0 的语法为基础。

路径表达式包含用斜杠（/）或双斜杠（//）字符隔开的一个或多个步骤。路径表达式可用步骤开头，也可用斜杠或双斜杠字符开头。最终步骤之前的每个步骤会生成节点序列，这些节点将用作后续步骤的上下文节点。

第一个步骤指定路径的起点，通常会通过使用返回节点或节点序列的函数调用或变量引用指定。初始“/”指示路径起始于包含上下文节点的树的根节点。初始“//”指示路径起始于初始节点序列，该序列由包含上下文节点的树的根节点及其所有后代组成。

每个步骤会重复执行，对上一步生成的每个上下文节点执行一次。重复执行的结果会合并以形成后续步骤的上下文节点序列。根据节点标识，此合并序列中的重复节点会被排除。

路径表达式的值是路径最终步骤产生的项的组合序列。此值可以是节点序列或原子值序列。因为路径中的每一步都为后续步骤提供了上下文节点，所以路径的最终步骤是可返回原子值序列的唯一步骤。同时返回节点和原子值的路径表达式会产生错误。

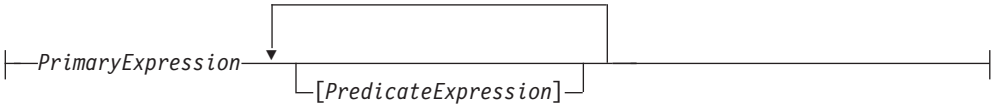
路径表达式产生的节点序列不能保证使用特定顺序。要了解路径表达式何时返回有序结果，请参阅描述 XQuery 表达式中结果的顺序的主题。

路径表达式的语法

路径表达式的每个步骤是轴步骤或过滤表达式。*轴步骤*返回可通过指定轴从上下文节点到达的节点的序列。*过滤表达式*由后跟零个或零个以上谓词的主表达式组成。

轴步骤:

过滤表达式:



/ 初始斜杠（/）指示路径起始于包含上下文节点的树的根节点，该节点必须是文档节点。路径表达式中的斜杠字符用于隔开步骤。

// 初始双斜杠字符（//）指示路径起始于初始节点序列，该序列由包含上下文节点的树的根节点及其所有后代组成，根节点必须是文档节点。要了解步骤之间的双斜杠的含义，请参阅有关缩写语法的主题。

axis
在 XML 文档或片段中移动的方向。受支持的轴列表中包括：子代、后代、属性、自身、后代或自身以及父代。某些轴可使用缩写语法表示。

node-test
对于轴步骤选择的每个节点必须为 true 的条件。此测试可以是根据节点名称选择节点的名称测试，或者是根据节点种类选择节点的种类测试。

PrimaryExpression
主表达式。

PredicateExpression
用于确定是保留还是废弃序列项的表达式。

示例

以下示例显示包括两个谓词的轴步骤。此步骤选择具有 secretary 子元素和 assistant 子元素的上下文节点的所有 employee 子代:

```
child::employee[secretary][assistant]
```

以下示例将过滤表达式用作路径表达式中的一个步骤。该表达式返回在给定书籍中包含多个脚注的每个章节或附录。

```
$book/(chapter | appendix)[fn:count(footnote)> 1]
```

轴步骤

*轴步骤*由三部分组成：*轴*（可选）、*节点测试*及零个或零个以上*谓词*。轴用来指定移动方向，节点测试指定用于选择节点的条件，而谓词用来过滤步骤返回的序列。

轴步骤的结果总是零个或零个以上节点的序列。

轴步骤可以是*正向步骤*或*逆向步骤*。正向步骤从上下文节点开始并且按文档顺序在 XML 树中移动，而逆向步骤从上下文节点开始并且按逆向文档顺序在 XML 树中移动。如果上下文项不是节点，那么表达式会产生错误。

轴步骤的非缩写语法包括用两个冒号隔开的轴名称和节点测试，并后跟零个或零个以上谓词。可通过省略轴并使用速记表示法来缩写轴表达式的语法。

在以下示例中，`child` 是轴的名称，而 `para` 是要对此轴选择的元素节点的名称。

```
child::para
```

此示例中的轴步骤选择充当上下文节点子代的所有 `para` 元素。

轴

*轴*是轴步骤的一部分，用于指定 XML 文档中的移动方向。

轴可以是正向轴或逆向轴。*正向轴*包含上下文节点和依据文档顺序在上下文节点后的节点。*逆向轴*包含上下文节点和依据文档顺序上下文节点之前的节点。

下表描述 DB2 XQuery 中受支持的轴。

表 16. DB2 XQuery 中受支持的轴

轴	描述	方向	注释
子代	返回上下文节点的子代。	正向	仅文档节点和元素节点具有子代。如果上下文节点是任何其他类型的节点，或者上下文节点是没有任何子代的文档节点或元素节点，那么子轴为空序列。文档节点或元素节点的子代可以是元素、处理指令、注释或文本节点。属性和文档节点决不能充当子代。
后代	返回上下文节点的后代（子代、子代的子代等等）。	正向	
属性	返回上下文节点的属性。	正向	如果上下文节点不是元素节点，那么此轴是空的。
自身	仅返回上下文节点。	正向	
后代-或-自身	返回上下文节点及其后代。	正向	
父代	返回上下文节点的父代，如果上下文节点没有父代，那么返回空序列。	逆向	即使属性节点决不能是元素节点的子代，但元素节点可以是属性节点的父代。

轴步骤选择节点序列时，将对每个节点指定一个上下文位置，该位置对应该节点在序列中的位置。如果该轴是正向轴，那么会按文档顺序对节点指定上下文位置，并且从 1 开始。如果该轴是逆向轴，那么会按逆向文档顺序对节点指定上下文位置，并且从 1 开始。指定上下文位置允许您通过选择位置从序列中选择节点。

节点测试

*节点测试*是一个条件，对于轴步骤选择的每个节点，此条件必须为 true。节点测试可表达为名称测试或种类测试。

*名称测试*根据节点名称选择节点。*种类测试*根据节点种类选择节点。

名称测试

名称测试由 QName 或通配符组成。在轴步骤中指定名称测试时，该步骤会在指定轴上选择与 QName 或通配符相匹配的节点。如果名称测试是对属性轴指定的，那么该步骤会选择与名称测试相匹配的所有属性。在所有其他轴上，该步骤会选择与名称测试相匹配的任何元素。如果在代码点方面节点的扩展 QName 等于在名称测试中指定的扩展 QName，那么表示 QName 相匹配。如果两个扩展 QName 的名称空间 URI 及局部名相等（即使名称空间前缀不相等），那么表示这两个扩展 QName 相等。

要点：在名称测试中指定的任何前缀必须对应于表达式的其中一个静态已知名称空间。对于在属性轴上执行的名称测试，无前缀 QName 没有名称空间 URI。对于在所有其他轴上测试的名称测试，无前缀 QName 具有缺省元素/类型名称空间的名称空间 URI。

下表描述在 DB2 XQuery 中受支持的名称测试。

表 17. DB2 XQuery 中受支持的名称测试

测试	描述	示例
QName	匹配其 QName 等于指定 QName 的任何节点（在指定轴上）。如果该轴为属性轴，那么此测试与属性节点相匹配。在所有其他轴上，此测试与元素节点相匹配。	在表达式 `child::para` 中，名称测试 `para` 会选择子轴上的所有 `para` 元素。
*	匹配指定轴上的所有节点。如果该轴为属性轴，那么此测试与所有属性节点相匹配。在所有其他轴上，此测试与所有元素节点相匹配。	在表达式 `child::*` 中，名称测试 * 与子轴上的所有元素相匹配。
NCName:*	指定表示 QName 的前缀部分的 *NCName*。此名称测试与指定轴上的某些节点相匹配，这些节点的名称空间 URI 与前缀绑定至的名称空间 URI 相匹配。如果该轴为属性轴，那么此测试与属性节点相匹配。在所有其他轴上，此测试与元素节点相匹配。	在表达式 `child::ns1:*` 中，名称测试 `ns1:*` 与子轴上的某些元素相匹配，这些元素与绑定至前缀 `ns1` 的名称空间相关联。
*:*NCName*	指定表示 QName 的局部的 *NCName*。此名称测试与指定轴上的某些节点相匹配，这些节点的局部名等于 *NCName*。如果该轴为属性轴，那么此测试与属性节点相匹配。在所有其他轴上，此测试与元素节点相匹配。	在表达式 `child::*:customerinfo` 中，名称测试 `*:customerinfo` 与子轴上局部名为 `customerinfo` 的所有元素相匹配，不管名称空间是否与元素名相关联都是如此。

种类测试

在轴步骤中指定种类测试时，该步骤仅在指定轴上选择与种类测试相匹配的节点。下表描述在 DB2 XQuery 中受支持的种类测试。

表 18. DB2 XQuery 中受支持的种类测试

测试	描述	示例
node()	匹配指定轴上的任何节点。	在表达式 child::node() 中，种类测试 node() 会选择子轴上的所有节点。
text()	匹配指定轴上的任何文本节点。	在表达式 child::text() 中，种类测试 text() 会选择子轴上的所有文本节点。
comment()	匹配指定轴上的任何注释节点。	在表达式 child::comment() 中，种类测试 comment() 会选择子轴上的所有注释节点。
processing-instruction()	匹配指定轴上的任何处理指令节点。	在表达式 child::processing-instruction() 中，种类测试 processing-instruction() 会选择子轴上的所有处理指令节点。
element() 或 element(*)	匹配指定轴上的任何元素节点。	在表达式 child::element() 中，种类测试 element() 会选择子轴上的所有元素节点。在表达式 child::element(*) 中，种类测试 element(*) 会选择子轴上的所有元素节点。
attribute() or attribute(*)	匹配指定轴上的任何属性节点。	在表达式 child::attribute() 中，种类测试 attribute() 会选择子轴上的所有属性节点。在表达式 child::attribute(*) 中，种类测试 attribute(*) 会选择子轴上的所有属性节点。
document-node()	匹配指定轴上的任何文档节点。	在表达式 self::document-node() 中，种类测试 document-node() 会选择充当上下文节点的文档节点。

路径表达式的缩写语法

XQuery 提供了缩写语法以表达路径表达式中的轴。

下表描述路径表达式中允许的缩写。

表 19. 路径表达式的缩写语法

缩写语法	描述	示例
未指定轴	child:: 的速记缩写，但轴步骤指定节点测试的 attribute() 时除外。轴步骤指定属性测试时，省略轴是 attribute:: 的速记形式。	路径表达式 section/para 是 child::section/child::para 的缩写。路径表达式 section/attribute() 是 child::section/attribute::attribute() 的缩写。
@	attribute:: 的速记缩写。	路径表达式 section/@id 是 child::section/attribute::id 的缩写。

表 19. 路径表达式的缩写语法 (续)

缩写语法	描述	示例
//	/descendant-or-self::node()/ 的速记缩写，但此缩写出现在路径表达式开头时除外。 此缩写出现在路径表达式开头时，轴步骤将选择初始节点序列，其中包含上下文节点所在的树的根节点，以及充当此根节点后代的所有节点。如果根节点不是文档节点，那么此表达式将返回错误。	路径表达式 div1//para 是 child::div1/descendant-or-self::node()/child::para 的缩写。
..	parent::node() 的速记缩写。	路径表达式 ../title 是 parent::node()/child::title 的缩写。

缩写语法和非缩写语法示例

下表提供缩写语法和非缩写语法的示例。

表 20. 非缩写语法和缩写语法

非缩写语法	缩写语法	结果
child::para	para	选择充当上下文节点子代的 para 元素。
child::*	*	选择充当上下文节点子代的所有元素。
child::text()	text()	选择充当上下文节点子代的所有文本节点。
child::node()	node()	选择上下文节点的所有子代。因为属性未被视作节点的子代，所以此表达式不返回任何属性节点。
attribute::name	@name	选择上下文节点的 name 属性。
attribute::*	@*	选择上下文节点的所有属性。
child::para[fn:position() = 1]	para[1]	选择充当上下文节点子代的第一个 para 元素。
child::para[fn:position() = fn:last()]	para[fn:last()]	选择充当上下文节点子代的最后一个 para 元素。
/child::book/child::chapter[fn:position() = 5] /child::section[fn:position() = 2]	/book/chapter[5]/section[2]	选择该书第 5 章第 2 节，其父代是包含上下文节点的文档节点。
child::para[attribute::type="warning"]	para[@type="warning"]	选择上下文节点的类型属性值为 warning 的所有 para 子代。
child::para[attribute::type='warning'] [fn:position() = 5]	para[@type="warning"][5]	选择上下文节点的类型属性值为 warning 的第 5 个 para 子代。

表 20. 非缩写语法和缩写语法 (续)

非缩写语法	缩写语法	结果
`child::para[fn:position() = 5] [attribute::type="warning"]`	`para[5][@type="warning"]`	选择上下文节点的类型属性值为 warning 的第 5 个 para 子代。
`child::chapter[child::title='Introduction']`	`chapter[title="Introduction"]`	选择上下文节点的 chapter 子代，该子代的一个或多个 title 子代的类型值等于字符串 Introduction。
`child::chapter[child::title]`	`chapter[title]`	选择上下文节点的 chapter 子代，该子代具有一个或多个 title 子代。

谓词

*谓词*会通过保留限定项来过滤序列。谓词由表达式组成，该表达式会用方括号（[]）括起来，称为谓词表达式。

谓词表达式会将所选项作为上下文项对序列中的每一项进行一次求值。谓词表达式的每次求值将返回 xs:boolean 值，称为*谓词布尔值*。谓词布尔值为 true 的项会保留下来，而谓词布尔值为 false 的项会被废弃。

下列规则用于确定谓词布尔值:

- 如果谓词表达式返回非数字值，那么谓词布尔值为谓词表达式的有效布尔值。
- 如果谓词表达式返回数字值，那么只有在序列中所处位置等于该数值的项的谓词布尔值才为 true。对于其他项，谓词布尔值为 false。此类谓词称为*数字谓词*或*位置谓词*。例如，在表达式 `$products[5]` 中，数字谓词 `[5]` 只保留绑定至变量 `$products` 的序列中的第 5 项。

  要点: 仅当序列具有确定顺序时，使用数字谓词从序列中选择的项才是确定的。

提示: 谓词的行为取决于谓词表达式是否返回数字值，这可能无法通过观察谓词表达式进行判断。可通过使用 fn:boolean 函数强制谓词使用有效布尔值，如 `[fn:boolean(PredicateExpression)]` 中所示。此外，还可通过使用 fn:position 函数强制谓词按位置谓词处理，如 `[fn:position() eq PredicateExpression]` 中所示。

下列示例具有谓词:

- `chapter[2]` 会选择作为上下文节点子代的第二个 chapter 元素。
- `descendant::toy[@color = "Red"]` 会选择充当元素 toy 并且颜色属性值为“Red”的所有上下文节点后代。
- `employee[secretary][assistant]` 会选择同时具有 secretary 子元素和 assistant 子元素的所有上下文节点职员子代。
- `(<cat />, <dog />, 47, <zebra />)[2]` 会返回元素 `<dog />`。

序列表达式

序列表达式用于构造、过滤和合并项目序列。序列不能嵌套。例如，将值 1、(2, 3) 和 ( ) 合并到单个序列中会产生序列 (1, 2, 3)。

用于构造序列的表达式

可通过使用逗号运算符或范围表达式来构造序列。

逗号运算符

要使用逗号运算符构造序列，请指定用逗号隔开的两个或更多操作数（表达式）。对序列表达式求值时，逗号运算符会对每个操作数求值，并将生成的序列并置以生成单个结果序列。例如，以下表达式将生成包含 5 个整数的序列:

```
(15, 1, 3, 5, 7)
```

序列可包含重复的原子值和节点。但是，序列决不能是另一个序列中的一项。如果新序列是通过并置两个或更多输入序列创建的，那么新序列将包含输入序列中的所有项，并且序列长度是输入序列长度的总和。

下列表达式使用逗号运算符来构造序列:

- 此表达式将长度分别为 1、2、0 和 2 的 4 个序列合并成长度为 5 的单个序列。此表达式将生成序列 10, 1, 2, 3, 4。

  ```
  (10, (1, 2), (), (3, 4))
  ```

- 此表达式将生成一个序列，包含所有充当上下文节点子代的 `salary` 元素（后跟充当上下文节点子代的所有 `bonus` 元素）。

  ```
  (salary, bonus)
  ```

- 假定变量 $price 绑定至值 10.50，那么此表达式将生成序列 10.50, 10.50。

  ```
  ($price, $price)
  ```

范围表达式

范围表达式会构造连续整数序列。范围表达式由通过 **to** 运算符隔开的两个操作数（表达式）组成。每个操作数的值必须可转换为类型为 xs:integer 的值。如果任一操作数为空序列，或者第一个操作数派生的整数大于第二个操作数派生的整数，那么范围表达式将生成空序列。否则会生成这样一个序列，该序列包含操作数派生的两个整数及其之间每个整数，这些整数按升序排列。例如，以下范围表达式求值为序列 1, 2, 3, 4:

```
(1 to 4)
```

以下示例使用范围表达式来构造序列:

- 以下示例在构造序列时将范围表达式用作一个操作数。序列表达式求值为序列 10, 1, 2, 3, 4。

  ```
  (10, 1 to 4)
  ```

- 以下示例构造长度为 1 并且包含单个整数 10 的序列。

  ```
  10 to 10
  ```

- 以下示例生成长度为 0 的序列。

  ```
  15 to 10
  ```

- 以下示例使用 fn:reverse 函数构造包含 6 个整数并且按降序排列的序列。此序列表达式求值为序列 15, 14, 13, 12, 11, 10。

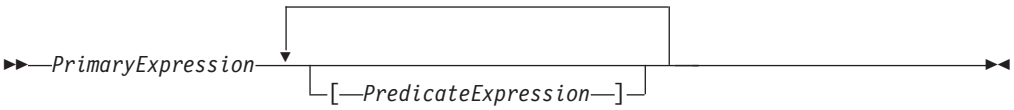
  ```
  fn:reverse(10 to 15)
  ```

过滤表达式

过滤表达式由后跟零个或零个以上谓词的主表达式组成。如果谓词存在，那么谓词用于过滤主表达式返回的序列。

过滤表达式的结果由谓词真实值为 ture 的所有项组成，这些项是由主表达式返回的。如果未指定任何谓词，那么结果就是主表达式的结果。结果序列中各项的排序与主表达式返回它们时的排序相同。在对谓词求值期间，每项都有一个上下文位置，用于表示它在该谓词正在过滤的序列中的位置。第一个上下文位置为 1。

语法



PrimaryExpression
主表达式。

PredicateExpression
用于确定是保留还是废弃序列项的表达式。

示例

以下示例使用过滤表达式返回过滤后的序列:

- 如果产品序列已绑定至变量，那么此表达式仅返回价格高于 100 的产品:

  ```
  $products[price gt 100]
  ```

- 以下表达式使用带有谓词的范围表达式来列示 1 到 100 之间可用 5 除的所有整数。因为范围表达式是用圆括号起来的，所以将作为主表达式处理:

  ```
  (1 to 100)[. mod 5 eq 0]
  ```

- 以下表达式生成整数 5:

  ```
  (1 to 21)[5]
  ```

- 以下表达式将过滤表达式用作路径表达式中的一个步骤。该表达式返回绑定至变量 $book 的书籍中的最后一章或附录:

  ```
  $book/(chapter | appendix)[fn:last()]
  ```

合并节点序列的表达式

DB2 XQuery 提供了一些运算符用于合并节点序列。这些运算符包括 **union**、**intersect** 和 **except**。

下表描述可用于合并节点序列的运算符。

表 21. 用于合并节点序列的 XQuery 运算符

运算符	描述
union 或 |	将两个节点序列当作操作数，并返回包含任一操作数中出现的所有节点的序列。**union** 关键字和 | 字符是等价的。
intersect	将两个节点序列当作操作数，并返回包含两个操作数中出现的所有节点的序列。
except	将两个节点序列当作操作数，并返回包含第一个操作数中出现但第二个操作数中未出现的所有节点的序列。

所有运算符会根据节点标识从结果序列中排除重复节点。生成的序列将按文档顺序返回。

union、**intersect** 或 **except** 的操作数必须解析为仅包含节点的序列。如果操作数包含并非节点的项，那么会返回错误。

除了本主题中描述的运算符以外，DB2 XQuery 还提供了一些函数以进行下列操作：对序列项或子序列进行索引访问（fn:index-of）、对序列中的项执行索引插入或除去操作（fn:insert-before 和 fn:remove）以及除去序列中的重复项（fn:distinct-values）。

示例

在这些示例中，假定变量 `$managers` 绑定至一组职员节点，这些节点表示身为经理的职员，并且变量 `$students` 也绑定至一组职员节点，这些节点表示身为学生的职员。

下列表达式都是有效示例，它们使用运算符组合节点序列：

- `$managers union $students` 会返回表示身为经理或学生的职员的一组节点。
- `$managers intersect $students` 会返回表示同时身为经理和学生的职员的一组节点。
- `$managers except $students` 会返回表示身为经理但不是学生的职员的一组节点。

算术表达式

算术表达式执行涉及加法、减法、乘法、除法和模数的运算。

下表按从高到低的运算符优先级顺序描述并列示算术运算符。除非使用圆括号强制对二目运算符进行求值，否则一元运算符的优先顺序高于二目运算符。

表 22. XQuery 中的算术运算符

运算符	目的	关联性
-(unary), +(unary)	对操作数的值求反，保留操作数的值	从右至左
*, div, idiv, mod	乘法，除法，除法求整，求模	从左至右
+, -	加法，减法	从左至右

注： 减法运算符之前必须加上空格，否则操作数可能解释为前一个标记的一部分。例如，`a-b` 解释为名称，但 `a - b` 和 `a -b` 解释为算术运算。

算术表达式的结果是数值、空序列或错误。对算术表达式求值时，每个操作数都被原子化（转换为原子值），并且存在下列规则：

- 如果原子化操作数为空序列，那么算术表达式的结果为空序列。
- 如果原子化操作数是包含多个值的序列，那么会返回错误。
- 如果原子化操作数是无类型的原子值（xdt:untypedAtomic），那么该值的数据类型将转换为 xs:double。如果强制类型转换失败，那么会返回错误。

如果操作数的类型在求值后变为算术运算符的有效组合，那么会对原子化操作数应用该运算符，并且此运算的结果是原子值或错误，例如，除零可能导致错误。如果操作数的类型不是算术运算符的有效组合，那么会返回错误。

表 23 标识算术运算符类型的有效组合。在下表中，字母 A 表示表达式中的第一个操作数，字母 B 表示第二个操作数。术语 numeric 指示类型 xs:integer、xs:decimal、xs:float、xs:double 或派生自其中一种类型的任何类型。如果运算符的结果类型列示为 numeric，那么结果类型将是有序列表（xs:integer、xs:decimal、xs:float 和 xs:double）中所有操作数通过子类型替换和类型提升转换可能形成的第一种类型。

表 23. 算术表达式操作数有效类型

带操作数的运算符	操作数 A 的类型	操作数 B 的类型	结果类型
A + B	numeric	numeric	numeric
A + B	xs:date	xdt:yearMonthDuration	xs:date
A + B	xdt:yearMonthDuration	xs:date	xs:date
A + B	xs:date	xdt:dayTimeDuration	xs:date
A + B	xdt:dayTimeDuration	xs:date	xs:date
A + B	xs:time	xdt:dayTimeDuration	xs:time
A + B	xdt:dayTimeDuration	xs:time	xs:time
A + B	xs:dateTime	xdt:yearMonthDuration	xs:dateTime
A + B	xdt:yearMonthDuration	xs:dateTime	xs:dateTime
A + B	xs:dateTime	xdt:dayTimeDuration	xs:dateTime
A + B	xdt:dayTimeDuration	xs:dateTime	xs:dateTime
A + B	xdt:yearMonthDuration	xdt:yearMonthDuration	xdt:yearMonthDuration
A + B	xdt:dayTimeDuration	xdt:dayTimeDuration	xdt:dayTimeDuration
A - B	numeric	numeric	numeric
A - B	xs:date	xs:date	xdt:dayTimeDuration
A - B	xs:date	xdt:yearMonthDuration	xs:date
A - B	xs:date	xdt:dayTimeDuration	xs:date
A - B	xs:time	xs:time	xdt:dayTimeDuration
A - B	xs:time	xdt:dayTimeDuration	xs:time
A - B	xs:dateTime	xs:dateTime	xdt:dayTimeDuration
A - B	xs:dateTime	xdt:yearMonthDuration	xs:dateTime
A - B	xs:dateTime	xdt:dayTimeDuration	xs:dateTime
A - B	xdt:yearMonthDuration	xdt:yearMonthDuration	xdt:yearMonthDuration
A - B	xdt:dayTimeDuration	xdt:dayTimeDuration	xdt:dayTimeDuration
A * B	numeric	numeric	numeric
A * B	xdt:yearMonthDuration	numeric	xdt:yearMonthDuration
A * B	numeric	xdt:yearMonthDuration	xdt:yearMonthDuration

表 23. 算术表达式操作数有效类型 (续)

带操作数的运算符	操作数 A 的类型	操作数 B 的类型	结果类型
A * B	xdt:dayTimeDuration	numeric	xdt:dayTimeDuration
A * B	numeric	xdt:dayTimeDuration	xdt:dayTimeDuration
A idiv B	numeric	numeric	xs:integer
A div B	numeric	numeric	numeric，但两个操作数都为 xs:integer 时则为 xs:decimal
A div B	xdt:yearMonthDuration	numeric	xdt:yearMonthDuration
A div B	xdt:dayTimeDuration	numeric	xdt:dayTimeDuration
A div B	xdt:yearMonthDuration	xdt:yearMonthDuration	xs:decimal
A div B	xdt:dayTimeDuration	xdt:dayTimeDuration	xs:decimal
A mod B	numeric	numeric	numeric

示例

- 在以下示例中，第一个表达式返回 xs:decimal 值 -1.5，而第二个表达式返回 xs:integer 值 -1:

```
-3 div 2
-3 idiv 2
```

- 在下面的表达式中，两个日期值的减法将生成类型为 xdt:dayTimeDuration 的值:

```
$emp/hiredate - $emp/birthdate
```

- 以下示例说明变量名称 `unit-price` 和 `unit-discount` 中使用的减号运算符与连字符之间的差别:

```
$unit-price - $unit-discount
```

比较表达式

比较表达式会对两个值进行比较。XQuery 提供了三种比较表达式: 值比较、常规比较和节点比较。

值比较

值比较会对两个原子值进行比较。值比较运算符包括 **eq**、**ne**、**lt**、**le**、**gt** 和 **ge**。

下表描述这些运算符。

表 24. XQuery 中的值比较运算符

运算符	目的
eq	如果第一个值等于第二个值，那么返回 true。
ne	如果第一个值不等于第二个值，那么返回 true。
lt	如果第一个值小于第二个值，那么返回 true。
le	如果第一个值小于或等于第二个值，那么返回 true。
gt	如果第一个值大于第二个值，那么返回 true。
ge	如果第一个值大于或等于第二个值，那么返回 true。

如果两个值的类型相同或者一个操作数是另一个操作数的子类型，那么可对两个值进行比较。可对数字类型（类型 xs:float、xs:integer、xs:decimal、xs:doubl 及派生自其中一种类型的类型）的两个操作数可进行比较。而且 xs:string 和 xs:anyURI 值可进行比较。

特殊值: 对于 xs:float 和 xs:double 值，正零和负零比较相等。INF 等于 INF，而 -INF 等于 -INF。NaN 不等于自身。正无穷大于所有其他非 NaN 值；负无穷小于所有其他非 NaN 值。NaN **ne** NaN 为 true，而涉及 NaN 值的任何其他比较为 false。如果类型为 xs:QName 的两个值的名称空间 URI 和局部名相等（名称空间前缀不重要），那么这两个值被视为相等。

值比较的结果可以是布尔值、空序列或错误。对值比较求值时，每个操作数都被原子化（转换为原子值），并且存在下列规则:

- 如果原子化操作数为空序列，那么值比较的结果为空序列。
- 如果原子化操作数是包含多个值的序列，那么会返回错误。
- 如果原子化操作数是无类型原子值（xdt:untypedAtomic），那么该值的数据类型将转换为 xs:string。

  将类型为 xdt:untypedAtomic 的值转换为 xs:string 使值比较变成可传递的。相反，常规比较遵循与转换无类型数据不同的规则，因此，常规比较是不可传递性的。在进行类型转换期间，值比较的传递性可能因为精度下降而受到损害。例如，因为 xs:float 的精度低于 xs:integer，所以有轻微差别的两个 xs:integer 值可能会被视为等于同一 xs:float 值。
- 如果操作数的类型在求值后对于运算符是有效组合，那么会对原子化操作数应用该运算符，并且比较结果将为 true 或 false。如果操作数的类型对于比较运算符是无效组合，那么会返回错误。

下列类型可使用 **eq** 或 **ne** 运算符进行比较。术语格列高利历指的是类型 xs:gYearMonth、xs:gYear、xs:gMonthDay、xs:gDay 和 xs:gMonth。对于接受两个格列高利历类型的操作数的二目运算符，两个操作数必须具有相同类型，例如，如果一个操作数的类型为 xs:gDay，那么另一个操作数的类型必须为 xs:gDay。术语数字指的是类型 xs:integer、xs:decimal、xs:float、xs:double 及派生自其中一种类型的任何类型。在进行涉及数值的比较时，会使用子类型替换和数字类型提升将操作数转换为有序列表（xs:integer、xs:decimal、xs:float 和 xs:double）中所有操作数都可转换至的第一种类型。

- 数字
- xs:boolean
- xs:string
- xs:date
- xs:time
- xs:dateTime
- xs:duration
- xdt:yearMonthDuration
- xdt:dayTimeDuration
- Gregorian
- xs:hexBinary
- xs:base64Binary

- xs:QName
- xs:NOTATION

下列类型可使用 **gt**、**lt**、**ge** 和 **le** 运算符进行比较。术语数字指的是类型 xs:integer、xs:decimal、xs:float 和 xs:double。在进行涉及数值的比较时，会使用子类型替换和数字类型提升将操作数转换为有序列表（xs:integer、xs:decimal、xs:float 和 xs:double）中所有操作数都可转换至的第一种类型。

- 数字
- xs:boolean
- xs:string
- xs:date
- xs:time
- xs:dateTime
- xdt:yearMonthDuration
- xdt:dayTimeDuration

示例

- 以下比较会使表达式 $book/author 返回的节点原子化。仅当原子化的结果为值“Kennedy”并且是 xs:string 或 xdt:untypedAtomic 的实例时，比较才为 true。如果原子化的结果是包含多个值的序列，那么会返回错误。

  ```
  $book1/author eq "Kennedy"
  ```

- 以下路径表达式包含谓词，用于选择重量超过 100 的产品。对于没有重量子元素的任何产品，该谓词的值为空序列，并且不会选择该产品:

  ```
  //product[weight gt 100]
  ```

- 因为在每种情况下两个构造节点在原子化后具有相同值（即使它们具有不同的标识或名称），所以下列比较返回 true:

  ```
  <a>5</a> eq <a>5</a>
  <a>5</a> eq <b>5</b>
  ```

常规比较

常规比较会比较任意长度的两个序列，以确定第一个序列中的至少一项和第二个序列中的一项是否符合指定的比较。常规比较运算符有 =、!=、<、<=、> 和 >=。

下表描述这些运算符。

表 25. XQuery 中的值比较运算符

运算符	目的
=	如果第一个序列中的某个值等于第二个序列中的某个值，那么返回 true。
!=	如果第一个序列中的某个值不等于第二个序列中的某个值，那么返回 true。
<	如果第一个序列中的某个值小于第二个序列中的某个值，那么返回 true。
<=	如果第一个序列中的某个值小于或等于第二个序列中的某个值，那么返回 true。
>	如果第一个序列中的某个值大于第二个序列中的某个值，那么返回 true。
>=	如果第一个序列中的某个值大于或等于第二个序列中的某个值，那么返回 true。

如后面的“示例”一节所示，常规比较不是可传递的，并且请注意，= 和 != 运算符不是相逆的。

常规比较的结果是布尔值或错误。对常规比较求值时，将对每个操作数进行原子化（转换为原子值序列）。比较各个原子值时，下列规则适用于所进行的隐式强制类型转换:

一个序列中的原子值	另一个序列中的原子值	将隐式类型值强制转换为的类型
xdt:untypedAtomic	数字类型	xs:double 如果您正在使用非常大的整数，那么可能会损失精度。例如，将 19 位数 9223372036854775672 强制转换为 xs:double 时，结果为 9.223372036854776E18（损失了三位精度）。通过将此值强制转换为 xs:decimal 或 xs:long 类型，即可避免产生此精度损失。
xdt:untypedAtomic	xdt:untypedAtomic 或 xs:string	xs:string
xdt:untypedAtomic	数字类型之外的值，xdt:untypedAtomic 或 xs:string	另一个值的类型

如果成功进行了强制类型转换，那么将使用下列其中一个值比较运算符来比较原子值: **eq**、**ne**、**lt**、**le**、**gt** 或 **ge**。如果有一对原子值，一个在第一个操作数序列中，另一个在第二个操作数序列中，并且这对原子值的比较为 true，那么比较结果为 true。例如，比较 (1, 2) = (2, 3) 将返回 true，原因是 2 eq 2 为 true。如果隐式强制类型转换操作失败，那么比较将返回 false。例如，以下语句中的比较 [b < 3.4] 将返回 false，原因是无法成功将字符串“N/A”强制类型转换为 xs:double:

```
Xquery let $doc := <a><b>N/A</b></a> return $doc[b < 3.4];
```

提示: 要逐项比较两个序列，请使用 XQuery 函数 fn:deep-equal。

示例

- 如果 $book1 的某个作者子元素的类型值为“Kennedy”，并且是 xs:string 或 xdt:untypedAtomic 的实例，那么以下比较为 true:

  ```
  $book1/author = "Kennedy"
  ```

- 以下示例包含三个常规比较。前两个比较的值为 true，而第三个比较的值为 false。此示例说明常规比较不可传递这一事实:

  ```
  (1, 2) = (2, 3)
  (2, 3) = (3, 4)
  (1, 2) = (3, 4)
  ```

- 以下示例包含两个常规比较，两个比较都为 true。此示例说明 = 和 != 运算符不是相逆的。

  ```
  (1, 2) = (2, 3)
  (1, 2) != (2, 3)
  ```

- 在以下示例中，变量 $a、$b 和 $c 已绑定至类型注释为 xdt:untypedAtomic 的元素节点。第一个元素节点包含字符串值“1”，第二个元素节点包含字符串值“2”，第三个元素节点包含字符串值“2.0”。在此示例中，以下表达式返回 false，原因是绑定至 $b 和 $c（“2”和“2.0”）的值是作为字符串进行比较的:

  ```
  ($a, $b) = ($c, 3.0)
  ```

  但是，以下表达式将返回 true，原因是绑定至 $b（“2”）的值与值 2.0 是作为数字进行比较的:

  ```
  ($a, $b) = ($c, 2.0)
  ```

节点比较

节点比较用于比较两个节点。可通过比较节点来确定它们是否共用同一标识，或者确定按文档顺序一个节点是否在另一个节点之前或之后。

下表描述 XQuery 中可用的节点比较运算符。

表 26. XQuery 中的节点比较运算符

运算符	目的
is	如果进行比较的两个节点具有相同标识，那么返回 true。
<<	如果按文档顺序第一个操作数节点在第二个操作数节点之前，那么返回 true。
>>	如果按文档顺序第一个操作数节点在第二个操作数节点之后，那么返回 true。

节点比较的结果是布尔值、空序列或错误。节点比较的结果是根据下列规则定义的:

- 每个操作数必须是单个节点或空序列；否则会返回错误。
- 如果任一操作数是空序列，那么比较的结果为空序列。
- 如果进行比较的两个节点具有同一标识，那么使用 **is** 运算符的比较为 true；否则比较为 false。
- 如果按文档顺序左操作数节点在右操作数节点之前，那么使用 << 运算符的比较返回 true；否则比较返回 false。
- 如果按文档顺序左操作数节点在右操作数节点之后，那么使用 >> 运算符的比较返回 true；否则比较返回 false。

示例

- 仅当左操作数和右操作数求值结果正好为同一节点时，以下比较才为 true:

  ```
  /books/book[isbn="1558604820"] is /books/book[call="QA76.9 C3845"]
  ```

- 以下比较为 false，原因是每个构造节点有自己的标识:

  ```
  <a>5</a> is <a>5</a>
  ```

- 仅当按文档顺序由左操作数标识的节点出现在由右操作数标识的节点之前时，以下比较才为 true:

  ```
  /transactions/purchase[parcel="28-451"] << /transactions/sale[parcel="33-870"]
  ```

逻辑表达式

逻辑表达式使用运算符 **and** 和 **or** 计算布尔值（true 或 false）。

下表按从高到低的运算符优先级顺序描述并列示算术运算符。

表 27. XQuery 中的逻辑表达式运算符

运算符	目的
and	如果两个表达式为 true，那么返回 true。
or	如果一个或两个表达式为 true，那么返回 true。

逻辑表达式的结果为布尔值（true 或 false）或错误。对逻辑表达式求值时，会确定每个操作数的有效布尔值（EBV）。然后会对操作数的 EBV 应用运算符，生成的结果是布尔值或错误。如果操作数的 EBV 是错误，那么逻辑表达式会产生错误。下表显示逻辑表达式根据其操作数的 EBV 返回的结果。

表 28. 根据操作数的 EBV 计算逻辑表达式的结果

操作数 1 的 EBV	运算符	操作数 2 的 EBV	结果
true	and	true	true
true	and	false	false
false	and	true	false
false	and	false	false
true	and	错误	错误
错误	and	true	错误
false	and	错误	false 或错误
错误	and	false	false 或错误
错误	and	错误	错误
true	or	true	true
false	or	false	false
true	or	false	true
false	or	true	true
true	or	错误	true 或错误
错误	or	true	true 或错误
false	or	错误	错误
错误	or	false	错误
错误	or	错误	错误

提示: 除逻辑表达式之外，XQuery 还提供了函数 fn:not，它会将常规序列作为参数并返回布尔值。

示例

- 下列表达式返回 true:

```
1 eq 1 and 2 eq 2
1 eq 1 or 2 eq 3
```

- 以下表达式可能返回 false 或错误:

```
1 eq 2 and 3 idiv 0 = 1
```

- 以下表达式可能返回 true 或错误:

```
1 eq 1 or 3 idiv 0 = 1
```

- 以下表达式返回错误:

```
1 eq 1 and 3 idiv 0 = 1
```

构造函数

构造函数会在查询中创建 XML 结构。XQuery 提供了构造函数以用于创建元素节点、属性节点、文档节点、文本节点、处理指令节点和注释节点。XQuery 提供了两种构造函数: 直接构造函数和计算构造函数。

*直接构造函数*使用类似 XML 的表示法在查询中创建 XML 结构。XQuery 提供了直接构造函数以用于创建元素节点（可能包括属性节点、文本节点和嵌套元素节点）、处理指令节点和注释节点。例如，以下构造函数会创建包含属性和某些嵌套元素的 book 元素:

```
<book isbn="isbn-0060229357">
  <title>Harold and the Purple Crayon</title>
  <author>
    <first>Crockett</first>
    <last>Johnson</last>
    </author>
</book>
```

*计算构造函数*使用基于带括号表达式的表示法在查询中使用 XML 结构。计算构造函数以标识要创建的节点类型的关键字开头，并且后跟节点的名称，然后是用于计算节点内容的带括号表达式。XQuery 提供了计算构造函数以用于创建元素节点、属性节点、文档节点、文本节点、处理指令节点和注释节点。例如，以下查询包含计算构造函数，它们生成的结果与前面示例中的直接构造函数完全相同:

```
element book {
  attribute isbn {"isbn-0060229357" },
  element title { "Harold and the Purple Crayon"},
  element author {
    element first { "Crockett" },
    element last {"Johnson" }
  }
}
```

构造函数中的带括号表达式

带括号表达式在构造函数中用于为元素和属性内容提供计算值。处理构造函数时会对这些表达式求值并替换为它们的值。带括号表达式是用花括号（{}）括起来的，以便与文字文本区别开来。

可在下列构造函数中使用带括号表达式以提供计算值:

- 直接元素构造函数:
  - 直接元素构造函数起始标记中的属性值可包含带括号表达式。
  - 直接元素构造函数的内容可包含带括号表达式，该表达式用于计算构造节点的内容和属性。
- 计算构造函数:
  - 可使用带括号表达式生成节点内容。

例如，以下直接元素构造函数包含带括号表达式:

```
<example>
  <p> Here is a query. </p>
  <eg> $b/title </eg>
  <p> Here is the result of the query. </p>
  <eg>{ $b/title }</eg>
</example>
```

对此构造函数求值时，它会产生以下结果（对此示例添加空格是为了提高可读性）:

```
<example>
  <p> Here is a query. </p>
  <eg> $b/title </eg>
  <p> Here is the result of the query. </p>
  <eg><title>Harold and the Purple Crayon</title></eg>
</example>
```

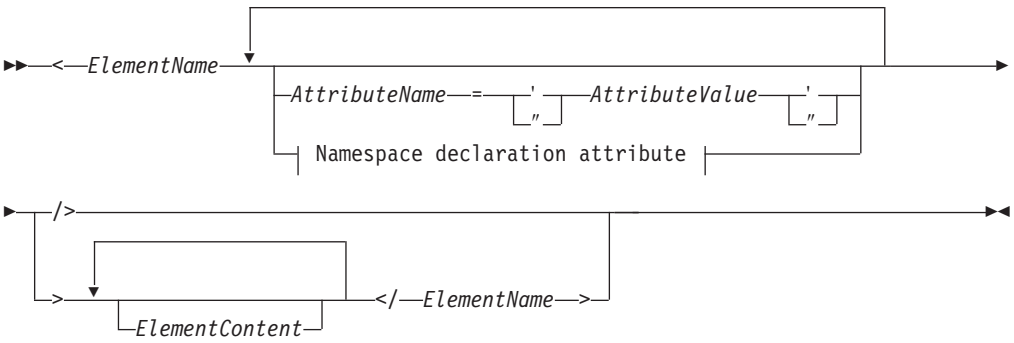
提示: 要在元素或属性内容中将花括号用作普通字符，可添加一对完全相同的花括号或使用字符引用。例如，可使用 {{ 对来表示字符 {。同样，可使用 }} 对来表示 }。此外，还可使用字符引用 { 和 } 来指示花括号字符。单个左花括号（{）被解释为带括号表达式的起始定界符。如果只有单个右花括号（}）而没有相匹配的左花括号，那么会发生错误。

直接元素构造函数

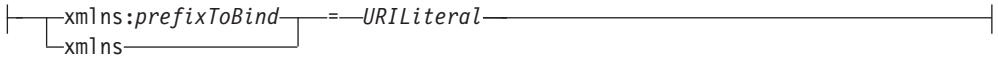
直接元素构造函数使用类似 XML 的表示法来创建元素节点。构造节点可以是简单元素或包含属性、文本内容和嵌套元素的复杂元素。

直接元素构造函数会生成生成具有自身节点标识的新元素节点。新元素节点的所有属性和后代节点同时也是具有其自身标识的新节点。

语法



Namespace declaration attribute:



ElementName

表示要构造的元素名称的 QName。在结尾标记中用于 *ElementName* 的名称必须与对应起始标记中使用的名称完全匹配，无论是否包括前缀都是如此。如果

ElementName 包括名称空间前缀，那么会通过使用静态已知名称空间将前缀解析为名称空间 URI。如果 *ElementName* 没有名称空间前缀，那么名称由缺省元素/类型名称空间隐式限定。通过对 *ElementName* 求值而生成的扩展 QName 将成为构造元素节点的名称。

AttributeName

表示要构造的属性名称的 QName。如果 *AttributeName* 包括名称空间前缀，那么会通过使用静态已知名称空间将前缀解析为名称空间 URI。如果 *AttributeName* 没有名称空间前缀，那么属性不会在任何名称空间中。通过对 *AttributeName* 求值而生成的扩展 QName 将成为构造属性节点的名称。每个属性的扩展 QName 必须是唯一的，否则表达式会产生错误。

直接元素构造函数中的每个属性都会创建新的属性节点及其自身节点标识。新属性节点的父代是构造元素节点。新属性节点的类型注释被指定为 xdt:untypedAtomic。

AttributeValue

用于指定属性值的字符串。属性值可包含带括号表达式（用花括号括起来的表达式），处理元素构造函数时会对这些表达式求值并替换为它们的值。预定义实体引用和字符引用也都有效，并且会替换为它们表示的字符。下表列示在 *AttributeValue* 中有效但必须用双重字符或实体引用表示的特殊字符。

表 29. 属性值中特殊字符的表示

字符	属性值中必需的表示
{	两个左花括号（{{）
}	两个右花括号（}}）
<	&lt;
&	&amp;
″	" 或两个双引号（″″）
’	' 或两个单引号（’’）

xmlns

名称空间声明属性的开始单词。在 QName 中指定为前缀时，**xmlns** 指示 *prefixToBind* 的值将绑定至由 *URILiteral* 指定的 URI。对于此构造函数表达式和嵌套在表达式中的所有表达式，此名称空间绑定将添加至静态识别名称空间，除非绑定被嵌套名称空间声明属性覆盖。在以下示例中，名称空间声明属性 `xmlns:metric = "http://example.org/metric/units"` 将前缀 `metric` 绑定至名称空间 `http://example.org/metric/units`。

指定为没有前缀的完整 QName 时，**xmlns** 指示缺省元素/类型名称空间设置为值 *URILiteral*。除非声明被嵌套名称空间声明属性覆盖，否则此缺省元素/类型名称空间对此构造函数表达式及其中嵌套的所有表达式有效。在以下示例中，名称空间声明属性 `xmlns = "http://example.org/animals"` 将缺省元素/类型名称空间设置为 `http://example.org/animals`。

prefixToBind

要绑定至为 *URILiteral* 指定的 URI 的前缀。*prefixToBind* 的值不能为 `xml` 或 `xmlns`。指定其中任意一个值会产生错误。

URILiteral

用于表示 URI 的字符串文字，是由用单引号或双引号括起来的零个或零个以上字符

组成的序列。字符串文字值必须为有效 URI。仅当名称空间声明属性用于设置缺省元素/类型名称空间时，*URILiteral* 的值才能为零长度字符串。否则，对 *URILiteral* 指定零长度字符串会产生错误。

ElementContent

直接元素构造函数的内容。该内容包含构造函数的起始标记与结尾标记之间的一切内容。序言中的边界空间声明将控制边界空间在元素构造函数中的处理方式。生成的内容序列是内容实体的并置。生成的所有相邻文本字符（包括带括号表达式生成的文本）将合并成单个文本节点。在生成的内容序列中，生成的所有属性节点必须在任何其他内容之前。

ElementContent 可由下列任意内容组成:

- **文本字符。**文本字符会创建文本节点，并且相邻文本节点会合并成单个文本节点。字符序列中的行结尾将根据对 XML 1.0 指定的行结束处理规则进行规范化。下表列示在 *ElementContent* 中有效但必须用双重字符或实体引用表示的特殊字符。

表 30. 元素内容中特殊字符的表示

字符	元素内容中必需的表示
{	两个左花括号（{{）
}	两个右花括号（}}）
<	&lt;
&	&amp;

- **嵌套直接构造函数。**
- **CDataSections。**CDataSections 是使用以下语法指定的: `<![CDATA[`*contents*`]]>`, 其中 *contents* 由一系列字符组成。对 *contents* 指定的字符（包括 < 和 & 之类的特殊字符）将被视作文字字符而不是定界符。序列]]> 用于终止 CDataSection, 因此不允许在 *contents* 中使用。
- **字符引用和预定义实体引用。**在处理期间，预定义实体引用和字符引用会扩展到被引用字符串中。
- **带括号表达式。**带括号表达式是用花括号括起来的 XQuery 表达式。例如，{5 + 7} 是一个带括号的表达式。带括号表达式的值可以是由节点和原子值组成的任意序列。带括号表达式可与直接元素构造函数的内容一起使用，以计算构造节点的内容和属性。对于带括号表达式返回的每个节点，将会创建该节点及其所有后代的新副本，而它们仍保留原始类型注释。*ElementContent* 返回的所有属性节点必须在生成的内容序列的开头；这些属性节点将成为构造元素的属性。*ElementContent* 返回的任何元素、内容或处理指令节点将成为新构造节点的子代。*ElementContent* 返回的所有原子值将转换为字符串并存储在文本节点中，而这些节点将成为构造节点的子代。相邻文本节点将合并成单个文本节点。

示例

- 以下直接元素构造函数会创建 book 元素。book 元素包含含有属性节点、一些嵌套元素节点和一些文本节点的复杂内容:

```
<book isbn="isbn-0060229357">
    <title>Harold and the Purple Crayon</title>
    <author>
```

```
        <first>Crockett</first>
        <last>Johnson</last>
      </author>
</book>
```

- 以下示例说明如何在直接元素构造函数中处理元素内容:
  - 以下表达式会构造具有一个子代的元素节点，该子代是包含值“1”的文本节点:
    ```
    <a>{1}</a>
    ```
  - 以下表达式会构造具有一个子代的元素节点，该子代是包含值“1 2 3”的文本节点:
    ```
    <a>{1, 2, 3}</a>
    ```
  - 以下表达式会构造具有一个子代的元素节点，该子代是包含值“123”的文本节点:
    ```
    <c>{1}{2}{3}</c>
    ```
  - 以下表达式会构造具有一个子代的元素节点，该子代是包含值“1 2 3”的文本节点:
    ```
    <b>{1, "2", "3"}</b>
    ```
  - 以下表达式会构造具有一个子代的元素节点，该子代是包含值“I saw 8 cats”的文本节点: :
    ```
    <fact>I saw 8 cats.</fact>
    ```
  - 以下表达式会构造具有一个子代的元素节点，该子代是包含值“I saw 8 cats”的文本节点:
    ```
    <fact>I saw {5 + 3} cats.</fact>
    ```
  - 以下表达式会构造具有以下三个子代的元素节点: 包含“I saw”的文本节点、名为 howmany 的子元素节点和包含"cats."的文本节点。子元素节点包含单个子代，即包含值“8”的文本节点。
    ```
    <fact>I saw <howmany>{5 + 3}</howmany> cats.</fact>
    ```

名称空间声明属性

名称空间声明属性是在直接元素构造函数中的起始标记中指定的。名称空间声明属性有两个用途: 将名称空间前缀绑定至 URI 以及为构造元素节点及其属性和后代设置缺省元素/类型名称空间。

从语法上说，名称空间声明属性的格式与直接元素构造函数中的属性相同: 属性是由名称和值指定的。属性名称是常量 QName。属性值是表示有效 URI 的字符串文字。

名称空间声明属性不会导致创建属性节点。

要点: 直接元素构造函数中每个名称空间声明属性的名称必须唯一，否则表达式会产生错误。

将名称空间前缀绑定至 URI

如果 QName 用后跟局部名的前缀 xmlns 开头，那么声明用于将名称空间前缀（指定为局部名）绑定至 URI（指定为属性值）。例如，名称空间声明属性 xmlns:metric = "http://example.org/metric/units" 将前缀 metric 绑定至名称空间 http://example.org/metric/units。

处理名称空间声明属性时，会将前缀及 URI 添加至构造函数表达式的静态已知名称空间，而新绑定会覆盖给定前缀的任何现有绑定。前缀和 URI 还会作为名称空间绑定添加至构造元素的名称空间作用域。

例如，在以下元素构造函数中，名称空间声明属性将用于绑定名称空间前缀 metric 和 english:

```
<box xmlns:metric = "http://example.org/metric/units"
xmlns:english = "http://example.org/english/units">
  <height> <metric:meters>3</metric:meters> </height>
  <width> <english:feet>6</english:feet> </width>
  <depth> <english:inches>18</english:inches> </depth>
</box>
```

设置缺省元素/类型名称空间

如果 QName 是没有前缀的 xmlns，那么声明用于设置缺省元素/类型名称空间。例如，名称空间声明属性 xmlns = "http://example.org/animals" 会将缺省元素/类型名称空间设置为 http://example.org/animals。

处理名称空间声明属性时，属性的值解释为名称空间 URI。此 URI 指定构造函数表达式的缺省元素/类型名称空间，而新的规范会覆盖任何现有缺省值。该 URI 还会添加至构造元素的名称空间作用域（不带前缀），而新的规范会覆盖不带前缀的任何现有名称空间绑定。如果名称空间 URI 是零长度字符串，那么构造函数表达式的缺省元素/类型名称空间会设置为“none”。

例如，在以下直接元素构造函数中，名称空间声明属性会将缺省元素/类型名称空间设置为 http://example.org/animals:

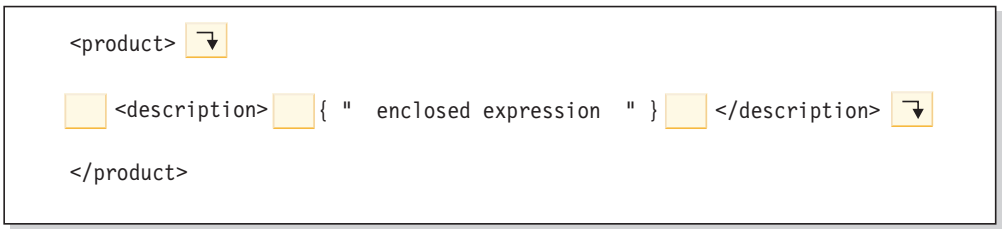
```
<cat xmlns = "http://example.org/animals">
  <breed>Persian</breed>
</cat>
```

直接元素构造函数中的边界空格

在直接元素构造函数中，*边界空格*是在每一端根据内容的开头或结尾、直接构造函数或带括号表达式定界的连续空格字符序列。

例如，可在构造函数的内容中使用边界空格将直接构造函数的结束标记与嵌套元素的起始标记隔开。

下图显示直接元素构造函数的示例，其边界空格将突出显示:



此示例中的边界空格包括下列字符：出现在 product 和 description 元素的起始标记之间的换行符和四个空格字符；出现在 description 元素的起始标记与带括号表达式之间的四个空格字符；出现在带括号表达式与 description 元素的结束标记之间的四个空格字符；以及出现在 description 元素的结束标记之后的一个换行符。

边界空格不包括下列任何类型的空格:

- 带括号表达式生成的空格
- 字符引用（例如 ）或 CDataSections 生成的字符
- 与字符引用 CDataSection 相邻的空格字符

边界空格策略控制元素构造函数是否保留边界空格。此策略是由查询序言中的边界空格声明指定的。如果边界空格声明指定 **strip**，那么会废弃边界空格。如果边界空格声明指定 **preserve**，那么会保留边界空格。如果未指定边界空格声明，那么缺省行为是在构造元素期间除去边界空格。

示例

- 在以下示例中，构造的 cat 元素节点有两个子元素节点，分别名为 breed 和 color:

```
<cat>
  <breed>{$b}</breed>
  <color>{$c}</color>
</cat>
```

  因为缺省情况下的边界空格策略为 **strip**，所以元素构造函数会除去子元素周围的空格。

- 在以下示例中，边界空格策略为 **strip**。此示例等价于 <a>abc:

```
declare boundary-space strip;
<a> {"abc"} </a>
```

- 但以下示例中的边界空格策略为 **preserve**。此示例等价于 <a> abc :

```
declare boundary-space preserve;
<a> {"abc"} </a>
```

  因为边界空格策略为 **preserve**，所以元素构造函数会保留出现在带括号表达式前后的空格。

- 在以下示例中，z 周围的空格并非边界空格。该空格将始终保留，并且此示例等价于 <a> z abc:

```
<a> z {"abc"}</a>
```

- 在以下示例中，不管边界空格策略如何，由字符引用生成的空格字符和相邻空格字符都会保留下来。此示例等价于 <a> abc:

```
<a>  &#x20;{"abc"}</a>
```

- 在以下示例中，不管边界空格策略如何，带括号表达式中的空格会保留下来，原因是带括号表达式生成的空格决不会被视为边界空格。此示例等价于 <a> :

```
<a>{" "}</a>
```

  带括号表达式中的两个空格会被元素构造函数保留下来，并出现在结果的起始标记与结束标记之间。

构造元素的名称空间作用域

构造元素节点的名称空间作用域属性由一组名称空间绑定组成。每个名称空间绑定使名称空间前缀与 URI 相关联。名称空间绑定会定义一组名称空间前缀，以用于在元素作用域中解释 QName。

要点: 要了解本主题，需要了解下列概念之间的差别:

静态已知名称空间

*静态已知名称空间*是表达式属性。此属性指示在处理表达式期间，XQuery 用于解析名称空间前缀的一组名称空间绑定。这些绑定未包括在查询结果中。

名称空间作用域

*名称空间作用域*是元素节点属性。此属性指示在处理元素及其内容时，可供

XQuery 外部的应用程序使用的一组名称空间绑定。这些绑定以序列化方式出现在查询结果中，所以它们可供外部应用程序使用。

构造元素的名称空间作用域包括使用下列方式创建的所有名称空间绑定:

- **通过名称空间声明属性显式进行。**将为下列构造函数中声明的每个名称空间声明属性创建一个名称空间绑定。
  - 当前元素构造函数。
  - 封闭直接元素构造函数（除非名称空间声明属性被当前元素构造函数或中间构造函数覆盖）。
- **由系统自动进行。**将在以下情况下创建名称空间绑定:
  - 要将前缀 `xml` 绑定至名称空间 URI `http://www.w3.org/XML/1998/namespace`。将为每个构造元素创建此绑定。
  - 对于构造元素名称或其属性名称中使用的每个元素（除非元素的名称空间作用域中已存在名称空间绑定）。如果节点的名称包含前缀，那么会在名称空间绑定中使用该前缀。如果该名称没有前缀，那么会为空前缀创建绑定。如果发生冲突（需要使用同一前缀的两个不同绑定），那么节点名中使用的前缀将更改为任意前缀，并且为该任意前缀创建名称空间绑定。

    切记: QName 中使用的前缀必须解析为有效 URI，或者该前缀的绑定不能添加至元素的名称空间作用域。如果不能解析 QName，那么表达式会产生错误。

示例

以下查询包含序言和主体，序言包含名称空间声明，而主体包含直接元素构造函数:

```
declare namespace p="http://example.com/ns/p";
declare namespace q="http://example.com/ns/q";
declare namespace f="http://example.com/ns/f";
<p:newElement q:b="{f:func(2)}" xmlns:r="http://example.com/ns/r"/>
```

序言中的名称空间声明会将名称空间绑定添加至表达式的静态已知名称空间。但是，仅当构造函数中的 QName 使用这些名称空间时，才会将名称空间绑定添加至构造元素的名称空间作用域。因此，`p:newElement` 的名称空间作用域包含下列名称空间绑定:

- `p = "http://example.com/ns/p"` - 此名称空间绑定将添加至名称空间作用域，原因是前缀 `p` 出现在 QName `p:newElement` 中。
- `q = "http://example.com/ns/q"` - 此名称空间绑定将添加至名称空间作用域，原因是前缀 `q` 出现在属性 QName `q:b` 中。
- `r = "http://example.com/ns/r"` - 此名称空间绑定将添加至名称空间作用域，原因是它是由名称空间声明属性定义的。
- `xml = "http://www.w3.org/XML/1998/namespace"` - 此名称空间绑定将添加至名称空间作用域，原因是它是对每个构造元素节点定义的。

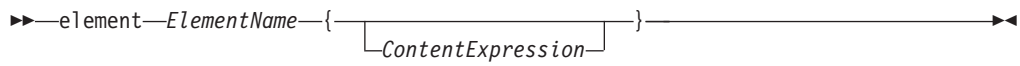
注意，名称空间 `f="http://example.com/ns/f"` 的任何绑定不会添加至名称空间作用域。这是因为元素构造函数未包含使用前缀 `f` 的元素或属性名，即使 `f:func(2)` 出现在属性 `q:b` 的内容中也是如此。因此，此名称空间绑定不会出现在查询结果中，即使它出现在静态已知名称空间中并且可在处理查询期间使用也是如此。

计算元素构造函数

计算元素构造函数会创建元素节点，其节点内容是根据带括号表达式计算的。

计算元素构造函数会生成具有自身节点标识的新元素节点。新元素节点的所有属性和后代节点同时也是具有其自身标识的新节点，即使它们是现有节点的副本亦如此。

语法



element
用于指示将构造元素节点的关键字。

ElementName
要构造的元素的 QName。如果 *ElementName* 包括名称空间前缀，那么会通过使用静态已知名称空间将前缀解析为名称空间 URI。如果 *ElementName* 没有名称空间前缀，那么名称由缺省元素/类型名称空间隐式限定。通过对 *ElementName* 求值而生成的扩展 QName 将成为构造元素节点的名称。

ContentExpression
用于生成构造元素节点内容的表达式。*ContentExpression* 的值可以是由节点和原子值组成的任何序列。*ContentExpression* 可用于计算构造节点的内容和属性。对于 *ContentExpression* 返回的每个节点，将会创建该节点及其所有后代的新副本，而它们仍保留原始类型注释。*ContentExpression* 返回的所有属性节点必须在节点序列的开头（在任何其他节点之前）；这些属性节点将成为构造元素的属性。*ContentExpression* 返回的任何元素、内容或处理指令节点将成为新构造节点的子代。*ContentExpression* 返回的所有原子值将转换为字符串并存储在文本节点中，而这些节点将成为构造节点的子代。相邻文本节点将合并成单个文本节点。

示例

在以下表达式中，计算元素构造函数将创建现有元素的修改副本。假定变量 `$e` 已绑定至具有数字内容的元素。此构造函数将创建新元素 `length`，它的属性与 `$e` 相同并且其数字内容是 `$e` 内容的两倍:

```
element length {$e/@*, 2 * fn:data($e)}
```

在此示例中，如果变量 `$e` 绑定至表达式 `let $e := <length units="inches">{5}</length>`，那么示例表达式将生成元素 `<length units="inches">10</length>`。

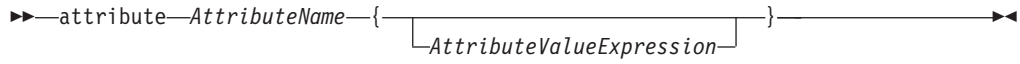
计算属性构造函数

计算属性构造函数会创建属性节点，其属性值是根据带括号表达式计算的。

计算属性构造函数会生成具有自身节点标识的新属性节点。

注: 要直接构造属性节点，请在直接元素构造函数中声明该属性。

语法



attribute
用于指示将构造属性节点的关键字。

AttributeName

要构造的属性的 QName。如果 *AttributeName* 包括名称空间前缀，那么会通过使用静态已知名称空间将前缀解析为名称空间 URI。如果 *AttributeName* 没有名称空间前缀，那么属性不会在任何名称空间中。通过对 *AttributeName* 求值而生成的扩展 QName 将成为构造属性节点的名称。元素中每个属性的扩展 QName 必须是唯一的，否则表达式会产生错误。

AttributeValueExpression

用于生成属性节点值的表达式。在处理期间，将对 *AttributeValueExpression* 的结果应用原子化，并且生成序列中的每个原子值的数据类型都将转换为字符串。通过强制类型转换生成的各个字符串将与插入的空格字符并置。并置字符串将成为构造属性节点的值。

示例

以下计算属性构造函数将构造名为 size 值为“7”的属性。

```
attribute size {4 + 3}
```

文档节点构造函数

所有文档节点构造函数都是计算构造函数。文档节点构造函数将创建文档节点，其节点内容是根据带括号表达式计算的。如果查询结果是完整的文档，文档节点构造函数会很有用。

文档节点构造函数会生成具有自身节点标识的新文档节点。

要点: 不会对构造文档节点执行任何验证。XQuery 文档节点构造函数不会强制执行用于控制 XML 文档结构的 XML 1.0 规则。例如，文档节点不必刚好有一个充当元素节点的子代。

语法

document

用于指示将构造文档节点的关键字。

ContentExpression

用于生成构造文档节点内容的表达式。*ContentExpression* 的值可以是由节点和原子值（属性节点除外）组成的任何序列。内容序列中的属性节点会产生错误。内容序列中的文档节点会替换为其子代。对于 *ContentExpression* 返回的每个节点，将会创建该节点及其所有后代的新副本，而它们仍保留原始类型注释。内容表达式返回的所有原子值将转换为字符串并存储在文本节点中，而这些节点将成为构造文档节点的子代。相邻文本节点将合并成单个文本节点。

示例

以下文档节点构造函数包括内容表达式，该表达式将返回包含根元素 `customer-list` 的 XML 文档:

```
document
{
<customer-list>
   {db2-fn:xmlcolumn('MYSCHEMA.CUSTOMER.INFO')/ns1:customerinfo/name}
</customer-list>
}
```

文本节点构造函数

所有文本节点构造函数都是计算构造函数。文本节点构造函数会创建文本节点，其节点内容是根据带括号表达式计算的。

文本节点构造函数会生成具有自身节点标识的新文本节点。

语法

text
用于指示将构造文本节点的关键字。

ContentExpression
用于生成构造文本节点内容的表达式。在处理期间，将对 *ContentExpression* 的结果应用原子化，并且生成序列中的每个原子值的数据类型都将转换为字符串。通过强制类型转换生成的各个字符串将与插入的空格字符并置。并置字符串将成为构造文本节点的内容。如果原子化产生空序列，那么不会构造文本节点。

注: 可使用文本节点构造函数来构造包含零长度字符串的文本节点。但是，如果在构造元素或文档节点的内容中使用此文本节点，那么会删除该文本节点或将其与另一文本节点合并到一起。

示例

以下构造函数创建包含字符串“Hello”的文本节点:

```
text {"Hello"}
```

处理指令构造函数

处理指令构造函数会创建处理指令节点。XQuery 提供了直接构造函数和计算构造函数以用于创建处理指令节点。

构造节点具有下列节点属性:

目标属性
标识处理指令将引导至的应用程序。

内容属性
指定处理指令的内容。

直接处理指令构造函数

直接处理指令构造函数使用类似 XML 的表示法来创建处理指令节点。

语法

PITarget
表示处理应用程序名称的 NCName，处理指令将被引导至该处理应用程序。处理指令的 PI 目标不能包含以任意大小写组合形式出现的字符“XML”。

DirPIContents
用于指定处理指令内容的一系列字符。处理指令的内容不能包含字符串 ?>。

示例

以下构造函数会创建处理指令节点:

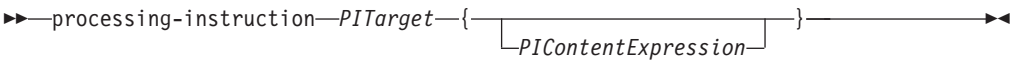
```
<?format role="output" ?>
```

计算处理指令构造函数

计算处理指令构造函数会创建处理指令节点，其节点内容是根据带括号表达式计算的。

计算处理指令构造函数会生成具有自身节点标识的新处理指令节点。

语法



processing-instruction
用于指示将构造处理指令节点的关键字。

PITarget
表示处理应用程序名称的 NCName，处理指令将被引导至该处理应用程序。此名称必须符合 *Namespaces in XML* 指定的 NCName 的格式。

PIContentExpression
用于生成处理指令节点内容的表达式。在处理期间，将对 *PIContentExpression* 的结果应用原子化，并且生成序列中的每个原子值的数据类型都将转换为字符串。通过强制类型转换生成的各个字符串将与插入的空格字符并置。前导空格将被除去，而并置字符串将成为处理指令节点的内容。如果原子化产生空序列，那么该序列会被替换为零长度字符串。内容序列不能包含字符串“?>”。

示例

以下计算构造函数创建处理指令 <?audio-output beep?>:

```
processing-instruction audio-output {"beep"}
```

注释构造函数

注释构造函数会创建注释节点。XQuery 提供了直接构造函数和计算构造函数以用于创建注释节点。

直接注释构造函数

直接注释构造函数使用类似 XML 的表示法来创建注释节点。

语法

DirCommentContents
用于指定注释内容的一系列字符。注释的内容不能包含两个连续的连字符或以连字符结尾。

示例

以下构造函数会创建注释节点:

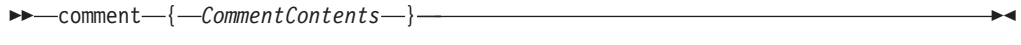
```
<!-- This is an XML comment. -->
```

计算注释构造函数

计算注释构造函数会创建注释节点，其节点内容是根据带括号表达式计算的。

计算注释构造函数会生成具有自身节点标识的新注释节点。

语法



comment
用于指示将构造注释节点的关键字。

CommentContents
用于生成注释内容的表达式。在处理期间，将对 *CommentContents* 的结果应用原子化，并且原子化序列中的每个原子值的数据类型都将转换为字符串。通过强制类型转换生成的各个字符串将与插入的空格字符并置。如果原子化产生空序列，那么该序列会被替换为零长度字符串。内容序列不能包含两个相邻连字符或以连字符结尾。

示例

以下计算构造函数创建注释 `<!--Houston, we have a problem.-->`:

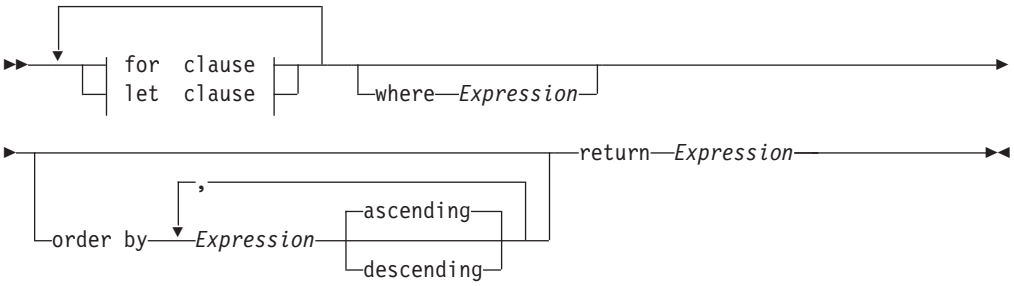
```
let $homebase := "Houston"
return comment {fn:concat($homebase, ", we have a problem.")}
```

FLWOR 表达式

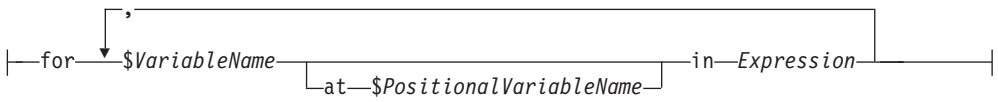
FLWOR 表达式会对序列进行迭代并将变量绑定至中间结果。FLWOR 表达式对于计算两个或更多文档之间的连接、重构数据和对结果排序非常有用。

FLWOR 表达式的语法

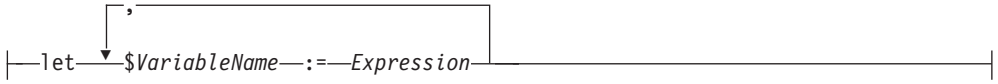
FLWOR 表达式由下列子句组成，其中一些子句是可选的: **for**、**let**、**where**、**order by** 和 **return**。



for clause:



let clause:



for

for 子句的起始关键字。**for** 子句会对 *Expression* 的结果进行迭代并将 *VariableName* 绑定至 *Expression* 返回的每一项。

let

let 子句的起始关键字。**let** 子句会将 *VariableName* 绑定至 *Expression* 的整个结果。

VariableName

变量的名称将绑定至 *Expression* 的结果。

PositionalVariableName

可选变量的名称，该变量绑定至某项的输入流中的某个位置，而该位置由 **for** 子句的每次迭代绑定。

Expression

XQuery 表达式。如果表达式包含顶级逗号运算符，那么必须用圆括号将表达式括起来。

where

where 子句的起始关键字。**where** 子句会过滤 **for** 和 **let** 子句生成的变量绑定元组。

order by

order by 子句的起始关键字。**order by** 子句会指定 **return** 子句处理值时使用的顺序。

ascending

指定按升序处理排序键。

descending
指定按降序处理排序键。

return
return 子句的起始关键字。**return** 子句中的表达式会对 **for**、**let**、**where** 和 **order by** 子句生成的每个绑定变量元组求值一次。如果 **return** 子句包含非更新表达式，那么 FLWOR 表达式为非更新表达式。对 return 子句求值的所有结果将并置到单个序列中，此序列是 FLWOR 表达式生成的。

如果 **return** 子句包含更新表达式，那么 FLWOR 表达式是更新表达式。必须在变换表达式的 **modify** 子句中指定更新 FLWOR 表达式。更新 FLWOR 表达式会生成更新列表。在将更新与变换表达式的 **modify** 子句中的其他更新表达式返回的其他更新合并后，包含变换的表达式会执行更新。

for 和 let 子句

FLWOR 表达式中的 **for** 或 **let** 子句会将一个或多个变量绑定至将在 FLWOR 表达式的其他子句中使用的值。

for 子句

for 子句会在表达式的结果中迭代，并将变量绑定至序列中的每项。

for 子句的最简单类型包含一个变量和关联表达式。在以下示例中，**for** 子句包含变量 $i 和用于构造序列 (1, 2, 3) 的表达式:

```
for $i in (1, 2, 3)
return <output>{$i}</output>
```

对 **for** 子句求值时，会创建三个变量绑定（序列中的每项对应一个绑定）:

```
$i = 1
$i = 2
$i = 3
```

示例中的 **return** 子句对每个绑定执行一次。该表达式会产生以下输出:

```
<output>1</output>
<output>2</output>
<output>3</output>
```

for 子句可包含多个变量，每个变量绑定至表达式的结果。在以下示例中，**for** 子句包含两个变量 $a 和 $b 及用于构造序列 1 2 和 4 5 的表达式:

```
for $a in (1, 2), $b in (4, 5)
return <output>{$a, $b}</output>
```

对 **for** 子句求值时，会为每个值组合创建一个变量绑定元组。这会产生四个变量绑定元组:

```
($a = 1, $b = 4)
($a = 2, $b = 4)
($a = 1, $b = 5)
($a = 2, $b = 5)
```

示例中的 **return** 子句会对每个绑定元组执行一次。该表达式会产生以下输出:

```
<output>1 4</output>
<output>2 4</output>
<output>1 5</output>
<output>2 5</output>
```

绑定表达式求值结果为空序列时，不会生成任何 **for** 绑定，也不会执行任何迭代。在以下示例中，绑定序列求值结果为空序列并且不执行任何迭代。不会返回 **return** 子句中的节点序列。

```
for $node in (<a test = "b" />, <a test = "c" />, <a test = "d" /> )[@test = "1"]
return
  <test>
    Sample return response
  </test>
```

for 子句中的位置变量

for 子句中绑定的每个变量都有一个同时绑定的关联位置变量。位置变量的名称会加上关键字 **at** 作为前缀。变量对序列中的各项进行迭代时，位置变量会对表示这些项在序列中的位置的整数（从 1 开始）进行迭代。

在以下示例中，**for** 子句包含变量 $cat 和用于构造序列 ("Persian", "Calico", "Siamese") 的表达式。该子句还包含位置变量 $i，属性构造函数中会引用该变量以计算 order 属性的值:

```
for $cat at $i in ("Persian", "Calico", "Siamese")
return <cat order = "{$i}"> { $cat } </cat>
```

对 **for** 子句求值时，会创建三个变量绑定元组，每个元组包含对应位置变量的绑定。

```
($i = 1, $cat = "Persian")
($i = 2, $cat = "Calico")
($i = 3, $cat = "Siamese")
```

示例中的 **return** 子句会对每个绑定元组执行一次。该表达式会产生以下输出:

```
<cat order = "1">Persian</cat>
<cat order = "2">Calico</cat>
<cat order = "3">Siamese</cat>
```

尽管每个输出元素都包含顺序属性，但并不能保证元素在输出流中的实际顺序，除非 FLWOR 表达式包含 order by $i 之类的 **order by** 子句。位置变量表示某个值在输入序列（而不是输出序列）中的序数位置。

let 子句

let 子句会将变量绑定至表达式的整个结果。let 子句不会执行任何迭代。

let 子句的最简单类型包含一个变量和关联表达式。在以下示例中，**let** 子句包含变量 $j 和用于构造序列 (1, 2, 3) 的表达式。

```
let $j := (1, 2, 3)
return <output>{$j}</output>
```

对 **let** 子句求值时，会为对表达式求值产生的整个序列创建单个绑定:

```
$j = 1 2 3
```

示例中的 **return** 子句执行一次。该表达式会产生以下输出:

```
<output>1 2 3</output>
```

let 子句可包含多个变量。但是，与 **for** 子句不同，**let** 子句会将每个变量绑定至其关联表达式的结果而不进行迭代。在以下示例中，**let** 子句包含两个变量 $a 和 $b 及用于构造序列 1 2 和 4 5 的表达式:

```
let $a := (1, 2), $b := (4, 5)
return <output>{$a, $b}</output>
```

对 **let** 子句求值时，会创建一个变量绑定元组:

```
($a = 1 2, $b = 4 5)
```

示例中的 **return** 子句对该元组执行一次。该表达式会产生以下输出:

```
<output>1 2 4 5</output>
```

绑定表达式求值结果为空序列时，会创建包含空序列的 let 绑定。

同一表达式中的 for 和 let 子句

如果 FLWOR 表达式同时包含 **for** 和 **let** 子句，那么 **let** 子句生成的变量绑定将添加至 **for** 子句生成的变量绑定。

在以下示例中，**for** 子句包含变量 $a 及用于构造序列 (1, 2, 3) 的表达式。**let** 子句包含变量 $b 和用于构造序列 (4, 5, 6) 的表达式:

```
for $a in (1, 2, 3)
let $b := (4, 5, 6)
return <output>{$a, $b}</output>
```

此示例中的 **for** 和 **let** 子句会产生三个绑定元组。元组数目由 **for** 子句确定。

```
($a = 1, $b = 4 5 6)
($a = 2, $b = 4 5 6)
($a = 3, $b = 4 5 6)
```

示例中的 **return** 子句会对每个绑定元组执行一次。该表达式会产生以下输出:

```
<output>1 4 5 6</output>
<output>2 4 5 6</output>
<output>3 4 5 6</output>
```

for 和 let 子句的比较

尽管 **for** 和 **let** 子句都会绑定变量，但它们绑定变量的方式有所不同。

下表提供了一些示例，它们会比较包含相似 **for** 和 **let** 子句的 FLWOR 表达式返回的结果。

*表 31. FLWOR 表达式中的 **for** 和 **let** 子句的比较*

查询描述	FLWOR 表达式	结果

使用 for 绑定单个变量	`for $i in ("a", "b", "c")`
	`return <output>{$i}</output>`
		`<output>a</output>`
		`<output>b</output>`
		`<output>c</output>`

使用 let 绑定单个变量	`let $i := ("a", "b", "c")` `return <output>{$i}</output>`	`<output>a b c</output>`

使用 for 绑定多个变量	`for $i in ("a", "b"), $j in ("c", "d")`
	`return <output>{$i, $j}</output>`
		`<output>a c</output>`
		`<output>b c</output>`
		`<output>a d</output>`
		`<output>b d</output>`

使用 let 绑定多个变量	`let $i := ("a", "b"), $j := ("c", "d")` `return <output>{$i, $j}</output>`	`<output>a b c d</output>`

注: 因为此表中的表达式未包括 **order by** 子句，所以输出元素的顺序是不确定的。

for 和 let 子句中的变量作用域

for 或 **let** 子句中绑定的变量在变量绑定后出现在 FLWOR 表达式的所有子表达式作用域中。

这意味着 **for** 或 **let** 子句可引用之前子句绑定的变量或同一子句之前绑定中的变量。

在以下示例中，FLWOR 表达式具有下列子句:

- 绑定变量 $orders 的 **let** 子句。
- 引用 $orders 并绑定变量 $i 的 **for** 子句。
- 引用 $orders 和 $i 并绑定变量 $c 的另一 **let** 子句。

该示例在一组订单中查找所有不同项目编号，并返回每个不同项目编号的订单数目。

```
let $orders := db2-fn:xmlcolumn("ORDERS.XMLORDER")
for $i in fn:distinct-values($orders/order/itemno)
let $c := fn:count($orders/order[itemno = $i])
return
<ordercount>
  <itemno> {$i} </itemno>
  <count> {$c} </count>
</ordercount>
```

要点: FLWOR 表达式的 **for** 和 **let** 子句不能多次绑定同一变量名。

where 子句

FLWOR 表达式中的 **where** 子句用于过滤由 **for** 和 **let** 子句生成的变量绑定元组。

where 子句用于指定应用于每个变量绑定元组的条件。如果条件为 true（即表达式生成有效布尔值 true），那么会保留该元组，并且会在执行 **return** 子句时使用它的绑定。否则会废弃该元组。

在以下示例中，**for** 子句会将变量 $x 和 $y 绑定至数值序列:

```
for $x in (1.5, 2.6, 1.9), $y in (.5, 1.6, 1.7)
where ((fn:floor($x) eq 1) and (fn:floor($y) eq 1))
return <output>{$x, $y}</output>
```

对 **for** 子句求值时，会创建 9 个变量绑定元组:

```
($x = 1.5, $y = .5)
($x = 2.6, $y = .5)
($x = 1.9, $y = .5)
($x = 1.5, $y = 1.6)
($x = 2.6, $y = 1.6)
($x = 1.9, $y = 1.6)
($x = 1.5, $y = 1.7)
($x = 2.6, $y = 1.7)
($x = 1.9, $y = 1.7)
```

where 子句会过滤这些元组，而下列元组会保留下来:

```
($x = 1.5, $y = 1.6)
($x = 1.9, $y = 1.6)
($x = 1.5, $y = 1.7)
($x = 1.9, $y = 1.7)
```

return 子句会对每个保留元组执行一次，而表达式会产生以下输出:

```
<output>1.5 1.6</output>
<output>1.9 1.6</output>
<output>1.5 1.7</output>
<output>1.9 1.7</output>
```

因为此示例中的表达式未包括 **order by** 子句，所以输出元素的顺序是不确定的。

order by 子句

FLWOR 表达式中的 **order by** 子句指定 **return** 子句处理值时所使用的顺序。如果没有 **order by** 子句，会按不确定的顺序返回 FLWOR 表达式的结果。

order by 子句包含一个或多个排序规范。排序规范用于对 **where** 子句过滤后保留下来的变量绑定元组重新排序。生成的排序将确定对 **return** 子句求值的顺序。

每个排序规范由表达式和顺序修饰符组成，会对表达式进行求值以生成排序键，而顺序修饰符用于指定排序键的排序（升序或降序）。两个元组的相对顺序通过将其排序键的值作为字符串进行比较来确定，处理顺序为从左至右。

在以下示例中，FLWOR 表达式包含 **order by** 子句，该子句会按照价格降序来对产品进行排序:

```
<price_list>{
    for $prod in db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')/product/description
    order by xs:decimal($prod/price) descending
    return
    <product>{$prod/name, $prod/price}</product>}
</price_list>
```

处理 **order by** 子句时，对于 **for** 子句生成的每个元组，会对排序规范中的表达式求值。对于第一个元组，表达式 xs:decimal($prod/price) 返回的值为 9.99。然后会针对下一个元组对表达式求值，而表达式会返回值 19.99。因为排序规范指示各项按降序排序，所以价格为 19.99 的产品会排在价格为 9.99 的产品之前。此排序过程会持续到所有元组的重新排序结束。然后，**return** 子句会对重新排序的元组流中的每个元组执行一次。

对 SAMPLE 数据库的 PRODUCT.DESCRIPTION 表运行时，示例中的查询将返回以下结果:

```
<price_list>
    <product>
        <name>Snow Shovel, Super Deluxe 26"</name>
        <price>49.99</price>
    </product>
    <product>
        <name>Snow Shovel, Deluxe 24"</name>
        <price>19.99</price></product>
    <product>
        <name>Snow Shovel, Basic 22"</name>
        <price>9.99</price>
    </product>
    <product>
        <name>Ice Scraper, Windshield 4" Wide</name>
        <price>3.99</price>
    </product>
</price_list>
```

在此示例中，排序规范中的表达式会根据 price 元素的值构造 xs:decimal 值。因为 price 元素的类型为 xdt:untypedAtomic，所以需要进行此类型转换。如果不进行转换，那么结果会使用字符串排序而不是数字排序。

提示: 可在 FLWOR 表达式中使用 **order by** 子句来指定本来不必迭代的查询中的值排序。例如，以下路径表达式会返回客户标识（Cid）高于 1000 的 customerinfo 元素列表:

```
db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo[@Cid > "1000"]
```

但是，要按客户名称的升序对这些项进行排序，您需要指定包含 **order by** 子句的 FLWOR 表达式:

```
for $custinfo in db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo
where ($custinfo/@Cid > "1000")
order by $custinfo/name ascending
return $custinfo
```

排序键不必包含在输出中。以下查询会生成产品名称列表并按价格降序排列，但输出中不包括价格:

```
for $prod in db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')/product
order by xs:decimal($prod/description/price) descending
return $prod/name
```

比较排序规范的规则

求值和比较排序规范的过程基于下列规则:

- 会对排序规范中的表达式进行求值并且对结果应用原子化。原子化的结果必须是单个原子值或空序列；否则会返回错误。对排序规范求值的结果称为排序键。
- 如果排序键的类型为 xdt:untypedAtomic，那么该键的数据类型会转换为 xs:string。如果一直将无类型值视作字符串，那么排序进程能够在不完全知道要排序的所有值类型的情况下开始。
- 如果排序规范生成的值并非同一类型，那么会通过子类型替换或类型提升将这些值（键）转换为公共类型。键是通过将键转换为支持 **gt** 运算符的最低公共类型来比较的。例如，如果排序规范生成包含 xs:anyURI 值和 xs:string 值的键列表，那么会使用类型为 xs:string 的 **gt** 运算符比较这些键。如果给定排序规范生成的排序键没有支持 **gt** 运算符的公共类型，那么会产生错误。
- 排序键的值用于确定将绑定变量元组传递至要执行的 return 子句时所使用的顺序。元组的排序是通过比较排序键确定的，处理顺序为从左至右，并且依据下列规则:
  - 如果排序顺序为升序，那么排序键较高的元组排在较后的位置。
  - 如果排序顺序为降序，那么排序键较高的元组排在较前的位置。

  排序键的大于关系的定义如下所示:
  - 空序列大于所有其他值。
  - NaN 解释为大于空序列以外的所有其他值。
  - 如果将某个值与另一个值进行比较时 **gt** 运算符返回 true，那么表示该值大于另一个值。
  - 特殊浮点值正零和负零无法比较大小，原因是 +0.0 gt -0.0 和 -0.0 gt +0.0 都为 false。

注: 如果指定了 **ascending** 选项（缺省值），那么排序键为空的元组出现在输出流的结尾，如果指定了 **descending** 选项，那么排序键为空的元组出现在输出流的开头。

return 子句

return 子句将生成 FLWOR 表达式的结果。

return 子句将对 FLWOR 表达式的其他子句生成的每个变量绑定元组进行一次求值。**return** 子句处理绑定变量元组时所使用的顺序是不确定的，除非 FLWOR 表达式包含 **order by** 子句。

如果 **return** 子句中的表达式是非更新表达式，那么所有 **return** 子句求值的结果将并置以形成非更新 FLWOR 表达式的结果。

如果 **return** 子句中的表达式是更新表达式，那么所有 **return** 子句求值的结果是更新列表。在将更新与变换表达式的 **modify** 子句中的其他更新表达式返回的更新合并后，包含 FLWOR 表达式的变换表达式会执行更新。

提示: 在 **return** 子句中，应使用圆括号将包含顶级逗号运算符的表达式括起来。因为 FLWOR 表达式的优先顺序高于逗号运算符的优先顺序，所以，如果不使用圆括号，包含逗号运算符的表达式可能会产生错误或意外结果。

FLWOR 示例

下列示例显示如何在完整查询中使用 FLWOR 表达式执行连接、分组和聚集。

用于连接 XML 数据的 FLWOR 表达式

以下查询会连接 SAMPLE 数据库的 PRODUCT 和 PURCHASEORDER 表中的 XML 数据，以列示 2005 年产生的订单中订购的产品名称。

因为产品文档和 PurchaseOrder 文档中的元素在同一名称空间中，所以查询会从声明缺省名称空间开始，这样查询中的元素名称就不需要前缀。特别是对于包含以“2005”开头的 OrderDate 属性值的订单，**for** 子句会对 PURCHASEORDER.PORDER 列进行迭代。对于每个订单，**let** 子句会将 partid 值指定给 *$parts* 变量。然后 **return** 子句会列示订单中包含的产品的名称。

```
for $po in db2-fn:xmlcolumn('PURCHASEORDER.PORDER')
  /PurchaseOrder[fn:starts-with(@OrderDate, "2005")]
let $parts := $po/item/partid
return
<ProductList PoNum = "{$po/@PoNum}">
    { db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')
    /product[@pid = $parts]/description/name }
</ProductList>
```

查询将返回以下结果:

```
<ProductList PoNum="5001">
    <name>Snow Shovel, Deluxe 24 inch</name>
    <name>Snow Shovel, Super Deluxe 26 inch</name>
    <name>Ice Scraper, Windshield 4 inch</name>
</ProductList>
<ProductList PoNum="5003">
    <name>Snow Shovel, Basic 22 inch</name>
</ProductList>
```

```
<ProductList PoNum="5004">
    <name>Snow Shovel, Basic 22 inch</name>
    <name>Snow Shovel, Super Deluxe 26 inch</name>
</ProductList>
```

用于对元素进行分组的 FLWOR 表达式

以下查询会按城市对 SAMPLE 数据库的 CUSTOMER 表中的客户名称进行分组。**for** 子句会对 customerinfo 文档进行迭代，并将每个城市元素绑定至变量 *$city*。对于每个城市，**let** 子句会将变量 *$cust-names* 绑定至该城市所有客户名称的无序列表。查询会返回城市元素，每个元素包含城市名称及居住在该城市的所有客户的嵌套名称元素。

```
for $city in fn:distinct-values(db2-fn:xmlcolumn('CUSTOMER.INFO')
    /customerinfo/addr/city)
let $cust-names := db2-fn:xmlcolumn('CUSTOMER.INFO')
    /customerinfo/name[../addr/city = $city]
order by $city
return <city>{$city, $cust-names} </city>
```

查询将返回以下结果:

```
<city>Aurora
    <name>Robert Shoemaker</name>
</city>
<city>Markham
    <name>Kathy Smith</name>
    <name>Jim Noodle</name>
</city>
<city>Toronto
    <name>Kathy Smith</name>
    <name>Matt Foreman</name>
    <name>Larry Menard</name>
</city>
```

用于聚集数据的 FLWOR 表达式

以下查询返回 2005 年的每张订单产生的总收益并创建 HTML 报告。

查询会对订单日期为 2005 年的每个 PurchaseOrder 元素进行迭代，并将该元素绑定至 **for** 子句中的变量 *$po*。然后，路径表达式 `$po/item/` 会将上下文位置移至 PurchaseOrder 元素中的每个项元素。嵌套表达式 `(price * quantity)` 会确定该项的总收益。fn:sum 函数会计算每项的总收益的生成序列之和。**let** 子句会将 fn:sum 函数的结果绑定至变量 *$revenue*。**order by** 子句会根据每张订单的总收益对结果进行排序。最后，**return** 子句会在报告表中为每张订单创建一行。

```
<html>
<body>
<h1>PO totals</h1>
<table>
<thead>
    <tr>
        <th>PO Number</th>
        <th>Status</th>
        <th>Revenue</th>
    </tr>
</thead>
<tbody>{
    for $po in db2-fn:xmlcolumn('PURCHASEORDER.PORDER')/
      PurchaseOrder[fn:starts-with(@OrderDate, "2005")]
    let $revenue := sum($po/item/(price * quantity))
    order by $revenue descending
    return
      <tr>
```

```
          <td>{string($po/@PoNum)}</td>
          <td>{string($po/@Status)}</td>
          <td>{$revenue}</td>
      </tr>
}
</tbody>
</table>
</body>
</html>
```

查询将返回以下结果:

```
<html>
<body>
<h1>PO totals</h1>
<table>
<thead>
    <tr>
        <th>PO Number</th>
        <th>Status</th>
        <th>Revenue</th>
    </tr>
</thead>
<tbody>
    <tr>
        <td>5004</td>
        <td>Shipped</td>
        <td>139.94</td>
    </tr>
    <tr>
        <td>5001</td>
        <td>Shipped</td>
        <td>123.96</td>
    </tr>
    <tr>
        <td>5003</td>
        <td>UnShipped</td>
        <td>9.99</td>
    </tr>
</tbody>
</table>
</body>
</html>
```

在浏览器中查看时，查询输出类似下表:

表 32. PO 总计

PO 编号	状态	收益
5004	已交付	139.94
5001	已交付	123.96
5003	未交付	9.99

用于更新 XML 数据的 FLWOR 表达式

以下示例使用 DB2 SAMPLE 数据库中的 CUSTOMER 表。在 CUSTOMER 表中，XML 列 INFO 包含客户地址和电话信息。

变换表达式会创建包含客户信息的 XML 文档的副本。在 **modify** 子句中，FLWOR 表达式和重命名表达式会将节点名 phone 的所有实例更改为名称 phonenumber:

```
xquery
transform
  copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1003')
  modify
    for $phone in $mycust/customerinfo/phone
    return
      do rename $phone as "phonenumber"
  return $mycust
```

对 SAMPLE 数据库运行时，该表达式会将节点名 phone 更改为 phonenumber 并返回以下结果:

```
<customerinfo Cid="1003">
  <name>Robert Shoemaker</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
  <phonenumber type="work">905-555-7258</phonenumber>
  <phonenumber type="home">416-555-2937</phonenumber>
  <phonenumber type="cell">905-555-8743</phonenumber>
  <phonenumber type="cottage">613-555-3278</phonenumber>
</customerinfo>
```

条件表达式

条件表达式使用关键字 **if**、**then** 和 **else** 以根据测试表达式的值是 true 还是 false 来对两个表达式的其中一个求值。

语法

if 直接放在测试表达式之前的关键字。

TestExpression
确定对哪部分条件表达式求值的 XQuery 表达式。

then
如果 *TestExpression* 的有效布尔值为 true，那么会对跟在此关键字之后的表达式求值。如果测试表达式的有效布尔值为 false，那么不会对表达式求值或检查是否有错。

else
如果 *TestExpression* 的有效布尔值为 false，那么会对跟在此关键字之后的表达式求值。如果测试表达式的有效布尔值为 true，那么不会对表达式求值或检查是否有错。

Expression
XQuery 表达式。如果表达式包含顶级逗号运算符，那么必须用圆括号将表达式括起来。

如果 **then** 或 **else** 条件分支包含更新表达式，那么条件表达式为更新表达式。更新表达式必须在变换表达式的 **modify** 子句中。

对于更新条件表达式，每个分支必须包含更新表达式或空序列。根据测试表达式的值，将选择 **then** 或 **else** 子句或对其求值。条件更新表达式将生成所选分支返回

的更新列表。在将更新与变换表达式的 **modify** 子句中的其他更新表达式返回的更新合并后，包含变换的表达式会执行更新。

示例

在以下示例中，查询会构造包含属性 basic 的 product 元素的列表。basic 属性的值是根据 price 元素的值是否小于 10 这一条件指定的:

```
for $prod in db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')/product/description
return (
if (xs:decimal($prod/price) < 10)
  then <product basic = "true">{fn:data($prod/name)}</product>
  else <product basic = "false">{fn:data($prod/name)}</product>)
```

查询将返回以下结果:

```
<product basic="true">Snow Shovel, Basic 22"</product>
<product basic="false">Snow Shovel, Deluxe 24"</product>
<product basic="false">Snow Shovel, Super Deluxe 26"</product>
<product basic="true">Ice Scraper, Windshield 4" Wide</product>
```

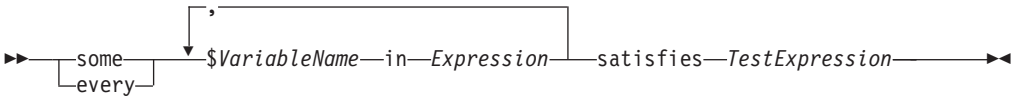
在此示例中，测试表达式会根据 price 元素的值构造 xs:decimal 值。xs:decimal 函数用于强制进行十进制比较。

定量表达式

如果一个或多个序列中的某项或每项都符合特定条件，那么定量表达式返回 true。定量表达式的值始终为 true 或 false。

定量表达式以量词（**some** 或 **every**）开头，用于指示表达式是执行现有量化还是全局量化。量词后跟一个或多个子句，以将变量绑定至表达式返回的项。然后会在测试表达式中引用绑定的变量，以确定是否某些或全部绑定值都符合特定条件。

语法



some

指定此关键字后，如果 *Expression* 返回的*至少一项*的 *TestExpression* 有效布尔值为 true，那么定量表达式返回 true。否则，定量表达式返回 false。

every

指定此关键字后，如果 *Expression* 返回的*每项*的 *TestExpression* 有效布尔值为 true，那么定量表达式返回 true。否则，定量表达式返回 false。

VariableName

要绑定至 *Expression* 结果中每一项的变量的名称。定量表达式中绑定的变量在定量表达式中绑定变量后出现的所有子表达式作用域中。

Expression

XQuery 表达式。如果表达式包含顶级逗号运算符，那么必须用圆括号将表达式括起来。

satisfies
直接放在测试表达式之前的关键字。

TestExpression
一个 XQuery 表达式，用于指定 *Expression* 返回的序列中的某项或每项必须符合的条件。

注: 发生错误时，定量比较的结果可能为布尔值或错误。

示例

- 如果 SAMPLE 数据库 CUSTOMER.INFO 列中的每个客户都有加拿大地址，那么以下示例中的定量表达式会返回 true:

```
every $cust in db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo
satisfies $cust/addr/@country = "Canada"
```

- 在下列示例中，对于绑定至变量 a 和 b 的值的每个组合（一共有 9 个组合），每个定量表达式会对其测试表达式求值。

  以下表达式的结果为 true:

```
some $a in (3, 5, 9), $b in (1, 3, 5)
satisfies $a * $b = 27
```

  以下表达式的结果为 false:

```
every $a in (3, 5, 9), $b in (1, 3, 5)
satisfies $a * $b = 27
```

- 以下示例说明定量表达式的结果不能确定是否存在错误。表达式可能返回 true 或错误，原因是测试表达式会对一个变量绑定返回 true，并对另一个变量返回错误:

```
some $a in (3, 5, "six") satisfies $a * 3 = 9
```

  同样，以下表达式可能返回 false 或错误:

```
every $a in (3, 5, "six") satisfies $a * 3 = 9
```

强制类型转换表达式

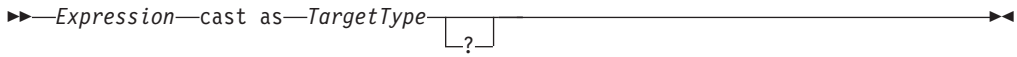
强制类型转换表达式会根据现有值创建特定类型的新值。

强制类型转换表达式有两个操作数: 输入表达式和目标类型。对强制类型转换表达式求值时，原子化用于将输入表达式的结果转换为原子值或空序列。如果原子化会生成包含多个原子值的序列，那么会返回错误。如果未返回错误，那么强制类型转换表达式会尝试根据输入值创建新的目标类型值。强制类型转换不支持输入和目标类型的某些组合。有关哪些类型可转换为其他类型的信息，请参阅第 21 页的『类型转换』。将值转换为某种数据类型时，可使用可转型表达式来测试该值是否能转换为该数据类型。

仅当目标类型后跟问号（？）时，空序列才是有效的输入值。

如果转型表达式的目标类型为 xs:QName 或派生自 xs:QName 或 xs:NOTATION 的类型，并且输入表达式类型为 xs:string 但不是文字串，那么会返回错误。

语法



Expression
返回单个原子值或空序列的 XQuery 表达式。仅当 *TargetType* 后跟问号（?）时，才允许使用空序列。

TargetType
Expression 的值将转换至的类型。*TargetType* 必须是属于预定义原子 XML 模式类型的原子类型。数据类型 xs:NOTATION、xdt:anyAtomicType 和 xs:anySimpleType 对于 *TargetType* 是无效类型。

? 指示 *Expression* 的结果可能为空序列。

示例

在以下示例中，强制类型转换表达式用于对 price 元素的值进行强制类型转换，该值将由类型 xs:string 转换为类型 xs:decimal。

```
for $price in db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')/product/description/price
return $price cast as xs:decimal
```

对 SAMPLE 数据库的 PRODUCT.DESCRIPTION 表运行时，示例中的查询将返回以下结果:

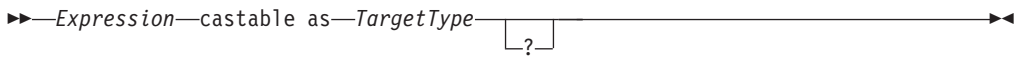
```
9.99
19.99
49.99
3.99
```

可转型表达式

可转型表达式测试某个值是否能转换为特定数据类型。如果该值可转换为该数据类型，那么可转型表达式返回 true。否则该表达式返回 false。

可转型表达式可用作谓词以避免求值时出现强制类型转换错误。还可使用它们来选择用于处理值的适当类型。有关哪些类型可转换为其他类型的信息，请参阅第 21 页的『类型转换』。

语法



Expression
返回单个原子值或空序列的 XQuery 表达式。

TargetType
用于测试 *Expression* 的值是否可进行强制类型转换的类型。*TargetType* 必须是属于预定义 XML 模式类型的原子类型。数据类型 xs:NOTATION、xdt:anyAtomicType 和 xs:anySimpleType 对于 *TargetType* 是无效类型。

? 指示空序列被视为有效的目标类型实例。如果 *Expression* 求值为空序列并且未指定 ?，那么可转型表达式返回 False。

返回的值

如果 *Expression* 可转型为 *TargetType*，那么可转型表达式返回 true。否则该表达式返回 false。

如果 *Expression* 的结果是空序列并且问号指示符跟在 *TargetType* 之后，那么可转型表达式返回 true。在以下示例中，问号指示符跟在目标类型 xs:integer 之后。

```
$prod/revision castable as xs:integer?
```

如果可转型表达式的 *TargetType* 为 xs:QName 或派生自 xs:QName 或 xs:NOTATION 的类型，并且 *Expression* 的类型为 xs:string 但并非文字串，那么可转型表达式的返回值为 false。

如果 *Expression* 的结果是包含多个原子值的序列，那么会返回错误。

示例

以下示例将可转型表达式用作谓词以避免求值时发生错误。以下示例避免 `@OrderDate` 并非有效日期时发生动态错误。

```
$po/orderID[if ( $po/@OrderDate castable as xs:date)
   then xs:date($po/@OrderDate) gt xs:date("2000-01-01")
   else false()]
```

仅当日期属性是 2000 年 1 月 1 日以后的有效日期时，谓词才会求值为 true 并返回 `orderID`。否则谓词求值为 false 并返回空序列。

以下示例使用可转型表达式来选择用于处理给定值的适当类型。该示例使用可转型表达式将邮政编码转换为整数或字符串:

```
if ($postalcode castable as xs:integer)
   then $postalcode cast as xs:integer
   else $postalcode cast as xs:string
```

以下示例在 FLWOR **let** 子句中使用可转型表达式测试 `$prod/mfgdate` 的值，并将值绑定至 `$currdate`。可转型表达式和转型表达式支持使用问号指示符处理空序列。

```
let $currdate := if ($prod/mfgdate castable as xs:date?)
   then $prod/mfgdate cast as xs:date?
   else "1000-01-01" cast as xs:date
```

如果 `$prod/mfgdate` 的值可转换为 xs:date，那么它会转换为该数据类型并绑定至 `$currdate`。如果 `$prod/mfgdate` 是空序列，那么会将空序列绑定至 `$currdate`。如果 `$prod/mfgdate` 不能转换为 xs:date，那么会将类型为 xs:date 的值 1000-01-01 绑定至 `$currdate`。

以下示例在执行比较之前使用可转型表达式测试产品类别的值。在 XML 列 FEATURES.INFO 中，文档包含元素 `/prod/category`。该值是数字代码或字符串代码。在执行比较之前，XMLEXISTS 谓词中的可转型表达式测试 `/prod/category` 的值以避免求值时发生错误。

```
SELECT F.PRODID FROM F FEATURES
WHERE xmlexists('$test/prod/category[ (( . castable as xs:double) and . > 100 ) or
   (( . castable as xs:string) and . > "A100" )]'
   passing F.INFO as "test")
```

返回的值是类别代码大于数字 100 或大于字符串“A100”的产品标识。

变换表达式和更新表达式

要使用 DB2 XQuery 更新现有 XML 数据，请使用变换表达式 **modify** 子句中的更新表达式。

在变换表达式中使用更新操作

必须在变换表达式的 **modify** 子句中使用 DB2 XQuery 更新表达式。更新表达式会作用于变换表达式 **copy** 子句创建的复制节点。

下列表达式是更新表达式:

- 删除表达式
- 插入表达式
- 重命名表达式
- 替换表达式
- 其 **return** 子句包含更新表达式的 FLWOR 表达式
- 其 **then** 或 **else** 子句包含更新表达式的条件表达式
- 用逗号隔开的两个或更多更新表达式，其中所有操作数都是更新表达式或空序列

DB2 XQuery 会对无效更新表达式返回错误。例如，如果条件表达式的一个分支包含更新表达式，而另一个分支包含并非空序列的非更新表达式，那么 DB2 XQuery 会返回错误。

变换表达式并非更新表达式，原因是它不会修改任何现有节点。变换表达式会创建现有节点的修改副本。变换表达式的结果可能包括通过更新变换表达式 **modify** 子句中的表达式创建的节点以及先前存在的节点。

处理 XQuery 更新操作

在变换表达式中，**modify** 子句可指定多个更新。例如，**modify** 子句可包含两个更新表达式，一个表达式用于替换现有值，另一个表达式用于插入新元素。**modify** 子句包含多个更新表达式时，每个更新表达式会独立求值，并生成由变换表达式 **copy** 子句创建的特定节点的关联更改操作列表。

在 **modify** 子句中，更新表达式不能修改由其他更新表达式添加的新节点。例如，如果一个更新表达式添加了新的元素节点，那么另一个更新表达式不能更改新创建节点的名称。

变换表达式 **modify** 子句中指定的所有更改操作会被收集到一起，并按以下顺序进行有效应用:

1. 下列更新操作是按不确定的顺序执行的:
   - 未使用 **before**、**after**、**as first** 或 **as last** 之类的排序关键字的插入操作。
   - 所有重命名操作。

- 替换操作，其中指定了关键字 **value of** 并且目标节点为属性、文本、注释或处理指令节点。

2. 使用 **before**、**after**、**as first** 或 **as last** 之类的排序关键字的插入操作。
3. 未指定关键字 **value of** 的替换操作。
4. 替换操作，其中指定了关键字 **value of** 并且目标节点为元素节点。
5. 所有删除操作。

应用更改操作的顺序应确保一系列多个更改将生成确定的结果。有关更新操作的执行顺序如何保证一系列多个更改生成确定结果的示例，请参阅『示例』中的最后一个 XQuery 表达式。

无效 XQuery 更新操作

处理变换表达式期间，如果出现下列其中一种情况，那么 DB2 XQuery 会返回错误:

- 对同一节点应用了两个或更多重命名操作。
- 对同一节点应用了使用关键字 **value of** 的两个或更多替换操作。
- 对同一节点应用了未使用关键字 **value of** 的两个或更多替换操作。
- 变换表达式的结果并非有效 XDM 实例。

  包含的元素的两个属性同名就是一个无效 XDM 实例的示例。
- XDM 实例包含不一致的名称空间绑定。

  下面是不一致的名称空间绑定的示例:
  - 属性节点的 QName 中的名称空间绑定与其父元素节点中的名称空间绑定不一致。
  - 具有同一父代的两个属性节点中的名称空间绑定相互不一致。

示例

在以下示例中，变换表达式的 **copy** 子句将变量 $product 绑定至元素节点的副本，而变换表达式的 **modify** 子句使用两个更新表达式来更改复制节点:

```
xquery
transform
copy $product := db2-fn:sqlquery(
  "select description from product where pid='100-100-01'")/product
modify(
  do replace value of $product/description/price with 349.95,
  do insert <status>Available</status> as last into $product )
return $product
```

以下示例在 SQL UPDATE 语句中使用 XQuery 变换表达式来修改 CUSTOMER 表中的 XML 数据。SQL UPDATE 语句会作用于 CUSTOMER 表的某行。变换表达式会根据该行的 INFO 列创建 XML 文档的副本，并将 status 元素添加至文档副本。UPDATE 语句会将该行 INFO 列中的文档替换为变换表达式修改后的文档副本:

```
UPDATE customer
SET info = xmlquery( 'transform
   copy $newinfo := $info
   modify do insert <status>Current</status> as last into $newinfo/customerinfo
   return $newinfo' passing info as "info")
WHERE cid = 1003
```

以下示例使用 DB2 SAMPLE 数据库中的 CUSTOMER 表。在 CUSTOMER 表中，XML 列 INFO 包含客户地址和电话信息。

在以下示例中，SQL SELECT 语句会作用于 CUSTOMER 表的某行。变换表达式的 **copy** 子句会根据列 INFO 创建 XML 文档的副本。删除表达式会删除文档副本中的地址信息和非工作电话号码。**return** 会使用 CUSTOMER 表的原始文档中的客户标识属性和国家或地区属性:

```
SELECT XMLQUERY( 'transform
  copy $mycust := $d
  modify
    do delete ( $mycust/customerinfo/addr,
       $mycust/customerinfo/phone[@type != "work"] )
  return
  <custinfo>
    <Cid>{data($d/customerinfo/@Cid)}</Cid>
    {$mycust/customerinfo/*}
    <country>{data($d/customerinfo/addr/@country)}</country>
  </custinfo>'
  passing INFO as "d")
FROM CUSTOMER
WHERE CID = 1003
```

对 SAMPLE 数据库运行时，该语句将返回以下结果:

```
<custinfo>
  <Cid>1003</Cid>
    <name>Robert Shoemaker</name>
    <phone type="work">905-555-7258</phone>
  <country>Canada</country>
</custinfo>
```

在以下示例中，XQuery 表达式会说明更新操作的顺序如何保证一系列多个更改将生成确定的结果。插入表达式会在 phone 元素后添加 status 元素，而替换表达式会将 phone 元素替换为 email 元素:

```
xquery
let $email  := <email>jnoodle@my-email.com</email>
let $status := <status>current</status>
return
  transform
  copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1002')
  modify (
    do replace $mycust/customerinfo/phone with $email,
    do insert $status after $mycust/customerinfo/phone[@type = "work"] )
  return $mycust
```

在 **modify** 子句中，替换表达式在插入表达式之前。但是，更新复制节点序列 $mycust 时，会在替换更新操作之前执行插入更新操作，以确保生成确定的结果。对 SAMPLE 数据库运行时，该表达式将返回以下结果:

```
<customerinfo Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
  <email>jnoodle@my-email.com</email>
  <status>current</status>
</customerinfo>
```

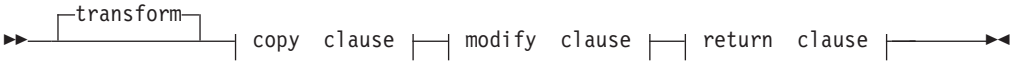
如果先执行替换操作，那么 phone 元素不会在节点序列中，而用于在 phone 元素后插入 status 元素的操作没有任何意义。

有关更新操作的顺序的信息，请参阅第 101 页的『处理 XQuery 更新操作』。

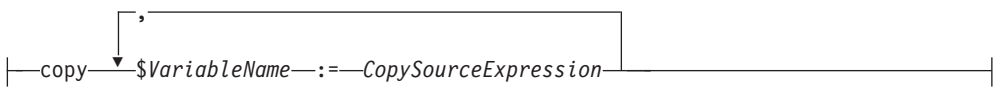
变换表达式

变换表达式会创建一个或多个节点的副本。变换表达式 **modify** 子句中的更新表达式会更改复制的节点。**return** 子句中的表达式会指定变换表达式的结果。

语法



copy clause:



modify clause:

return clause:

参数

transform
可用作变换表达式的可选起始关键字。

copy
变换表达式的 **copy** 子句的起始关键字。**copy** 子句中的每个 *VariableName* 绑定至相应 *CopySourceExpression* 所返回的节点树的逻辑副本。

VariableName
要绑定至 *CopySourceExpression* 返回的节点树的副本的变量名称。

CopySourceExpression
并非更新表达式的 XQuery 表达式。表达式必须返回单个节点及其称为*节点树*的后代（如果存在）。

如果表达式包含顶级逗号运算符，那么该表达式必须括在圆括号中。*CopySourceExpression* 求值时会作为元素构造函数中的带括号表达式处理。**copy** 子句创建的节点具有新的节点标识并且属于无类型节点。

modify
变换表达式的 **modify** 子句的起始关键字。

ModifyExpression
更新表达式或空序列。如果表达式包含顶级逗号运算符，那么该表达式必须括在圆括号中。将对更新表达式求值，并且会对 **copy** 子句创建的节点应用生成的更新。

如果更新表达式的目标节点并非包含变换表达式 **copy** 子句创建的节点，那么 DB2 XQuery 会返回错误。例如，如果重命名表达式尝试重命名 **copy** 子句未创建的节点，那么 DB2 XQuery 会返回错误。

modify 子句中指定的更新可能导致节点在其子代中有多个相邻文本节点。如果节点在其子代中有多个相邻文本节点，那么相邻文本节点将合并为单个文本节点。生成的文本节点的字符串值是没有插入空格的相邻文本节点的并置字符串值。如果创建的子节点是字符串值为零长度字符串的文本节点，那么会删除该文本节点。

return
变换表达式的 **return** 子句的起始关键字。

ReturnExpression
并非更新表达式的 XQuery 表达式。如果表达式包含顶级逗号运算符，那么该表达式必须括在圆括号中。

对 **return** 子句中的表达式求值并作为变换表达式的结果返回。**return** 子句中的表达式可访问通过在 **modify** 子句中更新表达式而更改或创建的节点。

变换表达式的 **return** 子句并未限制为仅返回 **copy** 子句创建的节点。*ReturnExpression* 可返回复制节点、原始节点和构造节点的任何组合。

示例

以下示例使用 DB2 SAMPLE 数据库中的 CUSTOMER 表。在 CUSTOMER 表中，XML 列 INFO 包含客户地址和电话信息。

在以下示例中，变换表达式的 **copy** 子句会根据列 INFO 创建 XML 文档的副本。在 **modify** 子句中，删除表达式会删除 XML 文档中电话的类型属性并非 home 的所有电话号码:

```
xquery
transform
   copy $mycust := db2-fn:sqlquery('select INFO from CUSTOMER where Cid = 1003')
   modify
     do delete $mycust/customerinfo/phone[@type!="home"]
   return $mycust;
```

对 SAMPLE 数据库运行时，该表达式将返回以下结果:

```
<customerinfo Cid="1003">
  <name>Robert Shoemaker</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
  <phone type="home">416-555-2937</phone>
</customerinfo>
```

以下表达式不会使用可选关键字 **transform**。变换表达式以 **copy** 子句开头并等价于先前的表达式。

```
xquery
copy $mycust := db2-fn:sqlquery('select INFO from CUSTOMER where Cid = 1003')
modify
   do delete $mycust/customerinfo/phone[@type!="home"]
return $mycust;
```

在以下示例中，SQL UPDATE 语句会修改并验证 CUSTOMER 表的某行中的 XML 文档。变换表达式的 **copy** 子句会根据列 INFO 创建 XML 文档的副本。替换表达式会更改 XML 文档副本中 name 元素的值。不会验证文档副本。XMLVALIDATE 函数会验证文档副本:

```
UPDATE customer set info = XMLVALIDATE(
  XMLQUERY('transform
  copy $mycust := $cust
  modify
    do replace value of $mycust/customerinfo/name with "Larry Menard, Jr."
  return $mycust'
  passing info as "cust" )
ACCORDING TO XMLSCHEMA ID customer)
where cid = 1005
```

基本更新表达式

通过使用 4 种基本 XQuery 更新表达式，可创建复杂更新表达式以更新现有 XML 数据。使用 DB2 XQuery 时，会在变换表达式的 **modify** 子句中使用更新表达式。

删除表达式

删除表达式会从节点序列中删除零个或零个以上节点。

语法

do delete
删除表达式的起始关键字。

TargetExpression
并非更新表达式的 XQuery 表达式。如果表达式包含顶级逗号运算符，那么该表达式必须括在圆括号中。*TargetExpression* 的结果必须为包含零个或零个以上节点的序列。每个节点的父代属性不能为空。

变换表达式会对删除表达式求值，并生成由要删除的节点组成的更新列表。与 *TargetExpression* 相匹配的任何节点将被标记为删除。删除 *TargetExpression* 节点时，这些节点会与其父节点断开连接。节点及节点子代不再包括在节点序列中。

示例

以下示例使用 DB2 SAMPLE 数据库中的 CUSTOMER 表。在 CUSTOMER 表中，XML 列 INFO 包含客户地址和电话信息。

以下表达式会从 XML 文档中删除地址元素及其所有子代节点，以及电话属性类型并非家庭的所有电话号码。

```
xquery
transform
copy $mycust := db2-fn:sqlquery('select INFO from CUSTOMER where Cid =1003')
modify
  do delete ($mycust/customerinfo/addr, $mycust/customerinfo/phone[@type!="home"])
return $mycust
```

对 SAMPLE 数据库运行时，该表达式将返回以下结果:

```
<customerinfo Cid="1003">
    <name>Robert Shoemaker</name>
    <phone type="home">416-555-2937</phone>
</customerinfo>
```

以下示例从任何电话元素节点中删除属性值为 home 的类型属性。

```
xquery
transform
copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1004')
modify (
  for $phone in $mycust/customerinfo//phone[@type="home"]
  return
    do delete $phone/@type )
return $mycust
```

对 SAMPLE 数据库运行时，该表达式将返回以下结果:

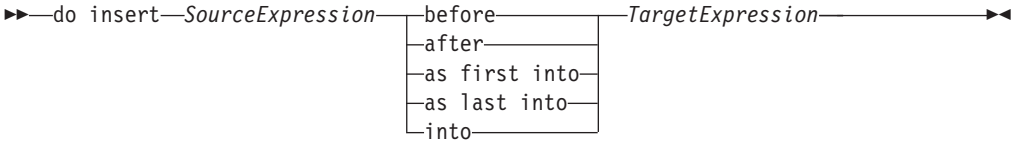
```
<customerinfo Cid="1004">
  <name>Matt Foreman</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M3Z 5H9</pcode-zip>
  </addr>
  <phone type="work">905-555-4789</phone>
  <phone>416-555-3376</phone>
  <assistant><name>Gopher Runner</name>
    <phone>416-555-3426</phone>
  </assistant>
</customerinfo>
```

该表达式会删除客户电话号码和助理电话号码中的类型属性。

插入表达式

插入表达式会将一个或多个节点的副本插入到节点序列的指定位置中。

语法



do insert
插入表达式的起始关键字。

SourceExpression
并非更新表达式的 XQuery 表达式。如果表达式包含顶级逗号运算符，那么必须用圆括号将表达式括起来。在元素构造函数中，会按带括号表达式的方式对该表达式

求值。*SourceExpression* 的结果为要插入的零个或零个以上节点的序列，称为*插入序列*。如果插入序列包含文档节点，那么插入序列中的文档节点将替换为其子代。

如果插入序列包含先出现在序列中的某些属性节点，那么这些属性会被添加至 *TargetExpression* 节点或其父代，这取决于指定的关键字。如果插入序列包含的某个属性节点跟在并非属性节点的节点之后，那么 DB2 XQuery 会返回错误。

before

用于指定 *SourceExpression* 节点的关键字将成为 *TargetExpression* 节点的前导同代节点。

SourceExpression 节点会直接插入到 *TargetExpression* 节点之前的位置。如果在 *TargetExpression* 之前插入多个节点，那么会按不确定的顺序插入它们，但已插入节点的集合会正好出现在 *TargetExpression* 之前。如果插入序列包含先出现在序列中的属性节点，那么这些属性节点会成为目标节点的父代。

after

用于指定 *SourceExpression* 节点的关键字将成为 *TargetExpression* 节点的后续同代节点。

SourceExpression 节点会直接插入到 *TargetExpression* 节点之后的位置。如果在 *TargetExpression* 中插入多个节点，那么会按不确定的顺序插入它们，但已插入节点的集合会正好出现在 *TargetExpression* 之后。如果插入序列包含先出现在序列中的属性节点，那么这些属性节点会成为目标节点的父代。

as first into

用于指定 *SourceExpression* 节点的关键字将成为 *TargetExpression* 节点的第一批子代。

如果多个节点作为 *TargetExpression* 节点的第一批子代插入，那么会按不确定的顺序插入它们，但已插入节点的集合会显示为 *TargetExpression* 的第一批子代。如果插入序列包含先出现在序列中的属性节点，那么这些属性节点会成为目标节点的属性。

as last into

用于指定 *SourceExpression* 节点的关键字将成为 *TargetExpression* 节点的最后一批子代。

如果多个节点作为 *TargetExpression* 节点的最后一批子代插入，那么会按不确定的顺序插入它们，但已插入节点的集合会显示为 *TargetExpression* 节点的最后一批子代。如果插入序列包含先出现在序列中的属性节点，那么这些属性节点会成为目标节点的属性。

into

用于指定 *SourceExpression* 节点的关键字将成为使用不确定顺序的 *TargetExpression* 节点的子代。

SourceExpression 节点将作为 *TargetExpression* 节点的子代插入到不确定的位置中。如果插入序列包含先出现在序列中的属性节点，那么这些属性节点会成为目标节点的属性。

TargetExpression

并非更新表达式的 XQuery 表达式。如果表达式包含顶级逗号运算符，那么必须用圆括号将表达式括起来。根据在 *TargetExpression* 之前指定的关键字，下列规则适用:

- 如果指定 **before** 或 **after**，那么 *TargetExpression* 的结果必须为父属性不为空的元素、文本、处理指令或注释节点。如果 *TargetExpression* 节点的父代是文档节点并且指定 **before** 或 **after**，那么插入序列不能包含属性节点。
- 如果指定 **into**、**as first into** 或 **as last into**，那么 *TargetExpression* 的结果必须为单个元素节点或单个文档节点。
- 如果指定 **into** 并且 *TargetExpression* 为文档节点，那么插入序列不能包含属性节点。

示例

以下示例使用 DB2 SAMPLE 数据库中的 CUSTOMER 表。在 CUSTOMER 表中，XML 列 INFO 包含客户地址和电话信息。

在以下示例中，变换表达式的 **copy** 子句会根据列 INFO 创建 XML 文档的副本。插入表达式会在最后一个 phone 元素之后插入 billto 元素及其所有子代:

```
xquery
   transform
   copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1004')
   modify
     do insert
       <billto country="Canada">
          <street>4441 Wagner</street>
          <city>Aurora</city>
          <prov-state>Ontario</prov-state>
          <pcode-zip>N8X 7F8</pcode-zip>
       </billto>
    after $mycust/customerinfo/phone[last()]
   return $mycust
```

对 SAMPLE 数据库运行时，该表达式将返回以下结果:

```
<customerinfo Cid="1004">
  <name>Matt Foreman</name>
    <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M3Z 5H9</pcode-zip>
  </addr>
  <phone type="work">905-555-4789</phone>
  <phone type="home">416-555-3376</phone>
  <billto country="Canada">
    <street>4441 Wagner</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </billto>
  <assistant>
    <name>Gopher Runner</name>
    <phone type="home">416-555-3426</phone>
  </assistant>
</customerinfo>
```

以下示例将值为 x2334 的属性 extension 插入到其类型属性为 work 的 phone 元素中:

```
  xquery
let $phoneext :=  attribute extension { "x2334" }
return
  transform
```

```
copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1004')
modify
  do insert  $phoneext into $mycust/*:customerinfo/*:phone[@type="work"]
return $mycust
```

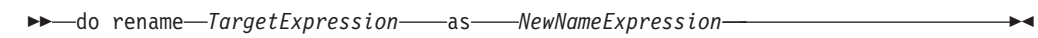
对 SAMPLE 数据库运行时，该表达式将返回以下结果:

```
<customerinfo Cid="1004">
  <name>Matt Foreman</name>
      <addr country="Canada">
      <street>1596 Baseline</street>
      <city>Toronto</city>
      <prov-state>Ontario</prov-state>
      <pcode-zip>M3Z 5H9</pcode-zip>
    </addr>
  <phone extension="x2334" type="work">905-555-4789</phone>
  <phone type="home">416-555-3376</phone>
  <assistant>
    <name>Gopher Runner</name>
    <phone type="home">416-555-3426</phone>
  </assistant>
</customerinfo>
```

重命名表达式

重命名表达式会将数据模型节点的名称属性替换为新的 QName。

语法



do rename
重命名表达式的起始关键字。

TargetExpression
并非更新表达式的 XQuery 表达式。*TargetExpression* 的结果必须是单个元素、属性或处理指令节点。如果表达式包含顶级逗号运算符，那么必须用圆括号将表达式括起来。

重命名表达式仅影响 *TargetExpression* 节点。如果 *TargetExpression* 节点是元素节点，那么表达式不会影响目标节点的任何属性或子代。如果 *TargetExpression* 节点是属性节点，那么表达式不会影响 *TargetExpression* 节点的父节点的其他属性或后代。

as 新名称表达式的起始关键字。

NewNameExpression
并非更新表达式的 XQuery 表达式。*NewNameExpression* 的结果必须是类型为 xs:string、xs:QName、xs:untypedAtomic 的值，或者原子化进程可从中抽取这类值的节点。如果表达式包含顶级逗号运算符，那么必须用圆括号将表达式括起来。

生成的值将转换为 QName，并根据静态已知名称空间解析其名称空间前缀（如果存在）。结果是错误或扩展 QName。扩展 QName 会替换 *TargetExpression* 节点的名称。

如果新的 QName 包含同一前缀但包含不同于 *TargetExpression* 节点的名称空间作用域的 URI，那么 DB2 XQuery 会返回错误。

示例

以下示例使用 DB2 SAMPLE 数据库中的 CUSTOMER 表。在 CUSTOMER 表中，XML 列 INFO 包含客户地址和电话信息。

在以下示例中，变换表达式的 **copy** 子句会根据列 INFO 创建 XML 文档的副本。重命名表达式会将 addr 元素的名称属性更改为 shipto:

```
xquery
transform
copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1000')
modify
  do rename $mycust/customerinfo/addr as "shipto"
return $mycust
```

对 SAMPLE 数据库运行时，该表达式将返回以下结果:

```
<customerinfo Cid="1000">
  <name>Kathy Smith</name>
  <shipto country="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </shipto>
  <phone type="work">416-555-1358</phone>
</customerinfo>
```

在以下示例中，变换表达式的 **modify** 子句包含 FLWOR 表达式和重命名表达式，后者会将元素 phone 的所有实例的名称属性更改为 phonenumber:

```
xquery
transform
copy $mycust :=  db2-fn:sqlquery('select info from customer where cid = 1003')
modify
  for $phone in $mycust/customerinfo/phone
  return
    do rename $phone as "phonenumber"
return $mycust
```

对 SAMPLE 数据库运行时，该表达式将返回以下结果:

```
<customerinfo Cid="1003">
  <name>Robert Shoemaker</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X 7F8</pcode-zip>
  </addr>
  <phonenumber type="work">905-555-7258</phonenumber>
  <phonenumber type="home">416-555-2937</phonenumber>
  <phonenumber type="cell">905-555-8743</phonenumber>
  <phonenumber type="cottage">613-555-3278</phonenumber>
</customerinfo>
```

在以下示例中，重命名表达式会将 addr 元素属性的名称从 country 更改为 geography:

```
xquery
transform
copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1000')
modify
  do rename $mycust/customerinfo/addr/@country as "geography"
return $mycust
```

对 SAMPLE 数据库运行时，该表达式将返回以下结果:

```
<customerinfo Cid="1000">
  <name>Kathy Smith</name>
  <addr geography="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type="work">416-555-1358</phone>
</customerinfo>
```

以下示例使用重命名表达式和 fn:QName 函数，以将名称空间前缀 other 添加至客户非工作电话号码元素和属性的名称。前缀 other 将绑定至 URI http://otherphone.com:

```
  xquery
transform
copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1004')
modify
  for $elem in $mycust/customerinfo/phone[@type != "work"]
  let $elemLocalName := fn:local-name($elem)
  let $newElemQName := fn:QName("http://otherphone.com", fn:concat("other:",
     $elemLocalName))
  return
    ( do rename $elem as $newElemQName,
    for $a in $elem/@* let $attrlocalname := fn:local-name($a)
    let $newAttrName := fn:QName("http://otherphone.com", fn:concat("other:",
       $attrlocalname))
    return
      do rename $a as $newAttrName )
return $mycust
```

对 SAMPLE 数据库运行时，该表达式将返回以下结果:

```
<customerinfo Cid="1004">
  <name>Matt Foreman</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M3Z 5H9</pcode-zip>
  </addr>
  <phone type="work">905-555-4789</phone>
  <other:phone xmlns:other="http://otherphone.com" other:type="home">
    416-555-3376</other:phone>
  <assistant>
    <name>Gopher Runner</name>
    <phone type="home">416-555-3426</phone>
  </assistant>
</customerinfo>
```

如果在变换表达式的 **return** 子句中使用以下表达式，那么使用缺省元素名称空间的 phone 元素节点会显示为 primary 节点的子节点，而 other:phone 元素节点会显示为 secondary 节点的子节点。

```
<phonenumbers xmlns:other="http://otherphone.com">
  <primary>
    { $mycust//phone }
  </primary>
  <secondary>
    { $mycust//other:phone }
  </secondary>
</phonenumbers>
```

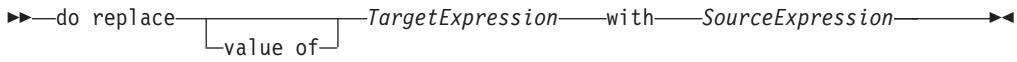
对 SAMPLE 数据库运行时，变换表达式将返回以下结果:

```
<phonenumbers xmlns:other="http://otherphone.com"
  <primary>
    <phone type="work">905-555-4789</phone>
    <phone type="home">416-555-3426</phone>
  </primary>
  <secondary>
    <other:phone other:type="home">416-555-3376</other:phone>
  </secondary>
</phonenumbers>
```

替换表达式

替换表达式会将现有节点替换为包含零个或零个以上节点的新序列，或在保留节点标识的同时替换节点值。

语法



do replace
替换表达式的起始关键字。

TargetExpression
并非更新表达式的 XQuery 表达式。如果表达式包含顶级逗号运算符，那么该表达式必须括在圆括号中。*TargetExpression* 的结果必须是并非文档节点的单个节点。如果 *TargetExpression* 的结果为文档节点，那么 DB2 XQuery 会返回错误。

如果未指定 **value of** 关键字，那么 *TargetExpression* 的结果必须是父属性不为空的单个节点。

value of
用于指定在替换 *TargetExpression* 节点值的同时保留节点标识的关键字。

with
源表达式的起始关键字。

SourceExpression
并非更新表达式的 XQuery 表达式。如果表达式包含顶级逗号运算符，那么该表达式必须括在圆括号中。

如果指定 **value of** 关键字，那么 *SourceExpression* 求值时会作为文本节点构造函数的内容表达式处理。*SourceExpression* 的结果是单个文本节点或空序列。

如果未指定 **value of** 关键字，那么 *SourceExpression* 的结果必须是节点序列。*SourceExpression* 在求值时会作为括在元素构造函数中的表达式处理。如果 *SourceExpression* 序列包含文档节点，那么文档节点会替换为其子代。*SourceExpression* 序列必须由下列节点类型组成:

- 如果 *TargetExpression* 节点是属性节点，那么替换序列必须由零个或零个以上属性节点组成。
- 如果 *TargetExpression* 节点是元素、文本、注释或处理指令节点，那么替换序列必须由零个或零个以上元素、文本、注释或处理指令节点的某些组合组成。

下列更新是在指定 **value of** 关键字后生成的:

- 如果 *TargetExpression* 节点是元素节点，那么 *TargetExpression* 节点的现有子代将替换为 *SourceExpression* 返回的文本节点。如果 *SourceExpression* 返回空序列，那么 *TargetExpression* 节点的子代属性将为空。如果 *TargetExpression* 节点包含属性节点，那么它们不受影响。
- 如果 *TargetExpression* 节点并非元素节点，那么 *TargetExpression* 节点的字符串值将替换为 *SourceExpression* 返回的文本节点的字符串值。如果 *SourceExpression* 未返回文本节点，那么 *TargetExpression* 节点的字符串值将替换为零长度字符串。

下列更新是在未指定 **value of** 关键字的情况下生成的:

- *SourceExpression* 节点会替换 *TargetExpression* 节点。*TargetExpression* 节点的父节点会成为每个 *SourceExpression* 节点的父代。*SourceExpression* 节点占据了节点层次结构中原来由 *TargetExpression* 节点占据的位置。
- *TargetExpression* 节点及其所有属性和后代将与节点序列分离。

示例

以下示例使用 DB2 SAMPLE 数据库中的 CUSTOMER 表。在 CUSTOMER 表中，XML 列 INFO 包含客户地址和电话信息。

在以下示例中，变换表达式的 **copy** 子句会根据列 INFO 创建 XML 文档的副本。替换表达式将替换 addr 元素及其子代:

```
xquery
transform
copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1000')
modify
  do replace $mycust/customerinfo/addr
    with
      <addr country="Canada">
        <street>1596 14th Avenue NW</street>
        <city>Calgary</city>
        <prov-state>Alberta</prov-state>
        <pcode-zip>T2N 1M7</pcode-zip>
      </addr>
return $mycust
```

对 SAMPLE 数据库运行时，该表达式将使用替换后的地址信息返回以下结果:

```
<customerinfo Cid="1000">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>1596 14th Avenue NW</street>
    <city>Calgary</city>
    <prov-state>Alberta</prov-state>
    <pcode-zip>T2N 1M7</pcode-zip>
  </addr>
    <phone type="work">416-555-1358</phone>
</customerinfo>
```

以下表达式会将客户电话元素 type 属性的值 home 替换为 personal:

```
  xquery
transform
copy $mycust := db2-fn:sqlquery('select info from customer where cid = 1004')
modify
  do replace value of $mycust/customerinfo/phone[@type="home"]/@type with "personal"
return $mycust
```

对 SAMPLE 数据库运行时，该表达式将使用替换后的属性值返回以下结果:

```
<customerinfo Cid="1004">
  <name>Matt Foreman</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M3Z 5H9</pcode-zip>
  </addr>
  <phone type="work">905-555-4789</phone>
  <phone type="personal">416-555-3376</phone>
  <assistant>
    <name>Gopher Runner</name>
    <phone type="home">416-555-3426</phone>
  </assistant>
</customerinfo>
```

助理的电话属性的值尚未更改。

第 5 章 内置函数

DB2 XQuery 提供用于处理 XML 数据的内置函数库。这些内置函数包括 XQuery 定义的函数和 DB2 内置函数。

XQuery 定义的函数

XQuery 定义的函数在绑定至前缀 `fn` 的名称空间中。此名称空间是缺省函数名称空间，这意味着可在未指定名称空间前缀的情况下调用 XQuery 定义的函数。如果使用查询序言中的缺省函数名称空间声明覆盖此缺省函数名称空间，那么必须使用前缀 `fn` 来调用 XQuery 定义的函数。

DB2 定义的函数

DB2 定义的函数包括 db2-fn:xmlcolumn 和 db2-fn:sqlquery，可使用它们访问 DB2 数据库中的 XML 值。前缀 `db2-fn` 并非缺省函数名称空间，所以在调用这些函数时必须使用名称空间前缀，除非使用查询序言中的缺省函数名称空间声明覆盖缺省名称空间。

按类别划分的 DB2 XQuery 函数

DB2 XQuery 函数的下列类别可用：字符串、布尔值、数字、日期和时间、序列、QName、节点及其他。

字符串函数

函数	描述
第 130 页的『codepoints-to-string 函数』	fn:codepoints-to-string 函数返回 Unicode 代码点序列的等价字符串。
第 130 页的『compare 函数』	fn:compare 函数比较两个字符串。
第 131 页的『concat 函数』	fn:concat 函数返回两个或更多原子值并置的字符串。
第 131 页的『contains 函数』	fn:contains 函数确定字符串是否包含特定子串。使用缺省整理来匹配搜索字符串。
第 141 页的『ends-with 函数』	fn:ends-with 函数确定字符串是否以特定子串结束。使用缺省整理来匹配搜索字符串。
第 150 页的『lower-case 函数』	fn:lower-case 函数将字符串转换为小写。
第 151 页的『matches 函数』	fn:matches 函数确定字符串是否与特定模式相匹配。
第 161 页的『normalize-space 函数』	fn:normalize-space 函数除去字符串中的前导和尾部空格字符，并将内部的每个空格字符序列替换为单个空白字符。
第 162 页的『normalize-unicode 函数』	fn:normalize-unicode 函数对字符串执行 Unicode 规范化。
第 166 页的『replace 函数』	fn:replace 函数将字符串中的每组字符与特定模式进行比较，然后将与该模式匹配的字符替换为另一组字符。

函数	描述
第 177 页的『starts-with 函数』	fn:starts-with 函数确定字符串是否以特定子串开头。使用缺省整理来匹配搜索字符串。
第 177 页的『string 函数』	fn:string 函数返回某个值的字符串表示。
第 178 页的『string-join 函数』	fn:string-join 函数返回通过并置各项并用分隔符进行分隔而生成的字符串。
第 179 页的『string-length 函数』	fn:string-length 函数返回字符串的长度。
第 179 页的『string-to-codepoints 函数』	fn:string-to-codepoints 函数返回对应于字符串值的 Unicode 代码点序列。
第 180 页的『substring 函数』	fn:substring 函数返回字符串的子串。
第 181 页的『substring-after 函数』	fn:substring-after 函数返回字符串中位于第一次出现的特定搜索字符串后面的子串。使用缺省整理来匹配搜索字符串。
第 182 页的『substring-before 函数』	fn:substring-before 函数返回字符串中位于第一次出现的特定搜索字符串前面的子串。使用缺省整理来匹配搜索字符串。
第 185 页的『tokenize 函数』	fn:tokenize 函数将字符串分为一系列子串。
第 187 页的『translate 函数』	fn:translate 函数将字符串中的所选字符替换为替换字符。
第 189 页的『upper-case 函数』	fn:upper-case 函数将字符串转换为大写。

布尔值函数

函数	描述
第 128 页的『boolean 函数』	fn:boolean 函数返回序列的有效布尔值。
第 143 页的『false 函数』	fn:false 函数返回 xs:boolean 值 false。
第 163 页的『not 函数』	如果序列的有效布尔值为 true，那么 fn:not 函数返回 false，如果序列的有效布尔值为 false，那么该函数返回 true。
第 188 页的『true 函数』	fn:true 函数返回 xs:boolean 值 true。

数字函数

函数	描述
第 126 页的『abs 函数』	fn:abs 函数返回数字值的绝对值。
第 127 页的『avg 函数』	fn:avg 函数返回序列中的平均值。
第 129 页的『ceiling 函数』	fn:ceiling 函数返回大于或等于特定数字值的最小整数。
第 143 页的『floor 函数』	fn:floor 函数返回小于或等于特定数字值的最大整数。
第 152 页的『max 函数』	fn:max 函数返回序列中的最大值。
第 153 页的『min 函数』	fn:min 函数返回序列中的最小值。
第 163 页的『number 函数』	fn:number 函数将值转换为 xs:double 数据类型。
第 170 页的『round 函数』	fn:round 函数返回最接近特定数字值的整数。

函数	描述
第 171 页的『round-half-to-even 函数』	fn:round-half-to-even 函数返回最接近特定数字值并具有指定精度的数字值。
第 183 页的『sum 函数』	fn:sum 函数返回序列中的值的总和。

日期、时间和持续时间函数

函数	描述
第 122 页的『adjust-date-to-timezone 函数』	fn:adjust-date-to-timezone 函数调整特定时区的 xs:date 值或除去该值的时区部分。
第 123 页的『adjust-dateTime-to-timezone 函数』	fn:adjust-dateTime-to-timezone 函数调整特定时区的 xs:dateTime 值或除去该值的时区部分。
第 125 页的『adjust-time-to-timezone 函数』	fn:adjust-time-to-timezone 函数调整特定时区的 xs:time 值或除去该值的时区部分。
第 132 页的『current-date 函数』	fn:current-date 函数返回 UTC 的隐式时区的当前日期。
第 133 页的『current-dateTime 函数』	fn:current-dateTime 函数返回 UTC 的隐式时区的当前日期和时间。
第 133 页的『current-local-date 函数』	db2-fn:current-local-date 函数返回本地时区的当前日期。
第 134 页的『current-local-dateTime 函数』	db2-fn:current-local-dateTime 函数返回本地时区的当前日期和时间。
第 134 页的『current-local-time 函数』	db2-fn:current-local-time 函数返回本地时区的当前时间。
第 134 页的『current-time 函数』	fn:current-time 函数返回 UTC 的隐式时区的当前时间。
第 135 页的『dateTime 函数』	fn:dateTime 函数根据 xs:date 值和 xs:time 值构造 xs:dateTime 值。
第 136 页的『day-from-date 函数』	fn:day-from-date 函数返回 xs:date 值的日期部分。
第 136 页的『day-from-dateTime 函数』	fn:day-from-dateTime 函数返回 xs:dateTime 值的日期部分。
第 137 页的『days-from-duration 函数』	fn:days-from-duration 函数返回持续日期的天数部分。
第 144 页的『hours-from-dateTime 函数』	fn:hours-from-dateTime 函数返回 xs:dateTime 值的小时数部分。
第 144 页的『hours-from-duration 函数』	fn:hours-from-duration 函数返回持续时间值的小时数部分。
第 145 页的『hours-from-time 函数』	fn:hours-from-time 函数返回 xs:time 值的小时数部分。
第 146 页的『implicit-timezone 函数』	fn:implicit-timezone 函数返回隐式时区值 PT0S，该值的类型为 xs:dayTimeDuration。值 PT0S 指示 UTC 为隐式时区。
第 149 页的『local-timezone 函数』	db2-fn:local-timezone 函数返回本地系统的时区。
第 154 页的『minutes-from-dateTime 函数』	fn:minutes-from-dateTime 函数返回 xs:dateTime 值的分钟数部分。

函数	描述
第 155 页的『minutes-from-duration 函数』	fn:minutes-from-duration 函数返回持续时间的分钟数部分。
第 156 页的『minutes-from-time 函数』	fn:minutes-from-time 函数返回 xs:time 值的分钟数部分。
第 156 页的『month-from-date 函数』	fn:month-from-date 函数返回 xs:date 值的月份部分。
第 157 页的『month-from-dateTime 函数』	fn:month-from-dateTime 函数返回 xs:dateTime 值的月份部分。
第 157 页的『months-from-duration 函数』	fn:months-from-duration 函数返回持续时间值的月数部分。
第 172 页的『seconds-from-dateTime 函数』	fn:seconds-from-dateTime 函数返回 xs:dateTime 值的秒数部分。
第 173 页的『seconds-from-duration 函数』	fn:seconds-from-duration 函数返回持续时间的秒数部分。
第 174 页的『seconds-from-time 函数』	fn:seconds-from-time 函数返回 xs:time 值的秒数部分。
第 184 页的『timezone-from-date 函数』	fn:timezone-from-date 函数返回 xs:date 值的时区部分。
第 184 页的『timezone-from-dateTime 函数』	fn:timezone-from-dateTime 函数返回 xs:dateTime 值的时区部分。
第 185 页的『timezone-from-time 函数』	fn:timezone-from-time 函数返回 xs:time 值的时区部分。
第 191 页的『year-from-date 函数』	fn:year-from-date 函数返回 xs:date 值的年份部分。
第 191 页的『year-from-dateTime 函数』	fn:year-from-dateTime 函数返回 xs:dateTime 值的年份部分。
第 192 页的『years-from-duration 函数』	fn:years-from-duration 函数返回持续时间的年数部分。

序列函数

函数	描述
第 132 页的『count 函数』	fn:count 函数返回序列中的若干值。
第 135 页的『data 函数』	fn:data 函数在将输入序列中的任何节点替换为其类型值后返回输入序列。
第 137 页的『deep-equal 函数』	fn:deep-equal 函数比较两个序列以确定它们是否符合深度相等的要求。
第 139 页的『distinct-values 函数』	fn:distinct-values 函数返回序列中的相异值。
第 140 页的『empty 函数』	fn:empty 函数指示自变量是否为空序列。
第 141 页的『exactly-one 函数』	如果自变量正好包含一项，fn:exactly-one 函数返回其自变量。
第 142 页的『exists 函数』	fn:exists 函数可以检查是否存在许多不同类型的项，例如，元素、属性、文本节点、静态值（例如，整数）或者 XML 文档。
第 148 页的『last 函数』	fn:last 函数返回序列中正在处理的值的数目。

函数	描述
第 146 页的『index-of 函数』	fn:index-of 函数返回某项出现在序列中的位置。
第 147 页的『insert-before 函数』	fn:insert-before 函数在一个序列中的特定位置之前插入另一序列。
第 164 页的『one-or-more 函数』	如果自变量包含一项或多项，那么 fn:one-or-more 函数返回其自变量。
第 165 页的『position 函数』	fn:position 函数返回序列中正在处理的上下文项的位置。
第 166 页的『remove 函数』	fn:remove 函数除去序列中的一项。
第 169 页的『reverse 函数』	fn:reverse 函数使序列中的项排序反向。
第 180 页的『subsequence 函数』	fn:subsequence 函数返回序列的子序列。
第 188 页的『unordered 函数』	fn:unordered 函数以不确定的顺序返回序列中的项。
第 193 页的『zero-or-one 函数』	如果自变量包含一项或为空序列，那么 fn:zero-or-one 函数返回自变量。

QName 函数

函数	描述
第 146 页的『in-scope-prefixes 函数』	fn:in-scope-prefixes 函数返回元素的所有名称空间作用域的前缀列表。
第 149 页的『local-name-from-QName 函数』	fn:local-name-from-QName 函数返回 xs:QName 值的局部。
第 160 页的『namespace-uri-for-prefix 函数』	fn:namespace-uri-for-prefix 函数返回与元素名称空间作用域中的前缀相关联的名称空间 URI。
第 160 页的『namespace-uri-from-QName 函数』	fn:namespace-uri-from-QName 函数返回 xs:QName 值的名称空间 URI 部分。
第 165 页的『QName 函数』	fn:QName 函数根据名称空间 URI 和包含词汇 QName（带有可选前缀）的字符串构建扩展名。
第 168 页的『resolve-QName 函数』	通过使用元素的名称空间作用域将名称空间前缀解析为名称空间 URI，fn:resolve-QName 函数将包含词汇 QName 的字符串转换为扩展 QName。

节点函数

函数	描述
第 148 页的『local-name 函数』	fn:local-name 函数返回节点的局部名属性。
第 158 页的『name 函数』	fn:name 函数返回节点名的前缀和局部名部分。
第 159 页的『namespace-uri 函数』	fn:namespace-uri 函数返回节点的限定名称的名称空间 URI。
第 161 页的『node-name 函数』	fn:node-name 函数返回节点的扩展 QName。
第 169 页的『root 函数』	fn:root 函数返回节点所属的树的根节点。

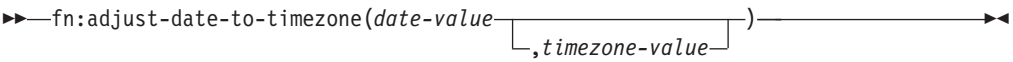
其他函数

函数	描述
第 139 页的『default-collation 函数』	fn:default-collation 函数返回表示表示为数据库定义的缺省整理的 URI。
第 174 页的『sqlquery 函数』	db2-fn:sqlquery 函数在当前相连 DB2 数据库中检索 SQL 全查询生成的序列。
第 190 页的『xmlcolumn 函数』	db2-fn:xmlcolumn 函数在当前相连 DB2 数据库中检索某列中的序列。

adjust-date-to-timezone 函数

fn:adjust-date-to-timezone 函数调整特定时区的 xs:date 值或除去该值的时区部分。

语法



date-value

要调整的日期值。

date-value 的类型为 xs:date，或者为空序列。

timezone-value

表示 *date-value* 要调整至的时区的持续时间。

timezone-value 可以是空序列或类型为 xdt:dayTimeDuration 的单值，范围在 -PT14H 到 PT14H 之间（包含两者）。该值可以是整数分钟值，并且不能有秒数部分。如果未指定 *timezone-value*，那么缺省值为 PT0H，表示 UTC。

返回的值

返回的值是类型为 xs:date 的值，或者是空序列，这取决于指定的参数。如果 *date-value* 不是空序列，那么返回的值类型为 xs:date。下表描述可能的返回值:

表 33. fn:adjust-date-to-timezone 的输入值和返回值的类型

date-value	*timezone-value*	返回的值
date-value，包含时区部分	显式值，或者未指定任何值（持续时间为 PT0H）	针对 *timezone-value* 表示的时区调整 *date-value*。
date-value，包含时区部分	空序列	*date-value* with no timezone component.
date-value，不包含时区部分	显式值，或者未指定任何值（持续时间为 PT0H）	*date-value*，带有时区部分。时区部分是由 *timezone-value* 表示的时区。未针对该时区调整日期部分。
date-value，不包含时区部分	空序列	*date-value*。
空序列	显式值、空序列或未指定任何值	空序列。

将 *date-value* 调整为另一时区时，*date-value* 被视为时间部分为 00:00:00 的 dateTime 值。返回的值包含 *timezone-value* 表示的时区部分。以下函数计算调整的当前日期值:

```
xs:date(fn:adjust-dateTime-to-timezone(xs:dateTime(date-value),timezone-value))
```

示例

在下列示例中，变量 `$tz` 为持续时间 -10 小时，定义为 `xdt:dayTimeDuration("-PT10H")`。

以下函数调整日期值 2002 年 5 月 7 日（UTC+1 时区）。该函数指定 *timezone-value* -PT10H。

```
fn:adjust-date-to-timezone(xs:date("2002-05-07+01:00"), $tz)
```

返回的日期值为 2002-05-06-10:00。日期调整为 UTC-10 时区。

以下函数将时区部分添加至日期值 2002 年 3 月 7 日（不带时区部分）。该函数指定 *timezone-value* -PT10H。

```
fn:adjust-date-to-timezone(xs:date("2002-03-07"), $tz)
```

返回的值为 2002-03-07-10:00。将对日期值添加时区部分。

以下函数调整日期值 2002 年 2 月 9 日（UTC-7 时区）。未指定 *timezone-value* 时，函数使用缺省时区 *timezone-value* PT0H。

```
fn:adjust-date-to-timezone(xs:date("2002-02-09-07:00"))
```

返回的日期为 2002-02-09Z，日期调整为 UTC。

以下函数除去日期值 2002 年 5 月 7 日（UTC-7 时区）的时区部分。*timezone-value* 是空序列。

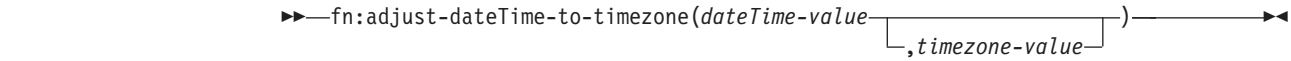
```
fn:adjust-date-to-timezone(xs:date("2002-05-07-07:00"), ())
```

返回的值是 2002-05-07。

adjust-dateTime-to-timezone 函数

fn:adjust-dateTime-to-timezone 函数调整特定时区的 xs:dateTime 值或除去该值的时区部分。

语法



dateTime-value

要调整的 dateTime 值。

dateTime-value 的类型为 xs:dateTime，或者是空序列。

timezone-value

表示 *dateTime-value* 要调整至的时区的持续时间。

timezone-value 可以是空序列或类型为 xdt:dayTimeDuration 的单值，范围在 -PT14H 到 PT14H 之间（包含两者）。该值可以是整数分钟值，并且不能有秒数部分。如果未指定 *timezone-value*，那么缺省值为 PT0H，表示 UTC。

返回的值

返回的值是类型为 xs:dateTime 的值，或者是空序列，这取决于输入值的类型。如果 *dateTime-value* 不是空序列，那么返回的值类型为 xs:dateTime。下表描述可能的返回值:

表 34. fn:adjust-dateTime-to-timezone 的输入值和返回值的类型

dateTime-value	*timezone-value*	返回的值
dateTime-value，包含时区部分	显式值，或者未指定任何值（持续时间为 PT0H）	*dateTime-value* 调整为 *timezone-value* 表示的时区。返回的值包含 *timezone-value* 表示的时区部分。
dateTime-value，包含时区部分	空序列	*dateTime-value*，没有时区部分。
dateTime-value，不包含时区部分。	显式值，或者未指定任何值（持续时间为 PT0H）	*dateTime-value*，带有时区部分。时区部分是由 *timezone-value* 表示的时区。日期和时区部分未调整为该时区。
dateTime-value，不包含时区部分。	空序列	*dateTime-value*。
空序列	显式值、空序列或未指定任何值	空序列。

示例

在下列示例中，变量 `$tz` 为持续时间 -10 小时，定义为 `xdt:dayTimeDuration("-PT10H")`。

以下函数将 dateTime 2007 年 3 月 7 日上午 10 点（UTC-7 时区）调整为 *timezone-value* -PT10H 指定的时区。

```
fn:adjust-dateTime-to-timezone(xs:dateTime("2002-03-07T10:00:00-07:00"), $tz)
```

返回的 dateTime 为 2002-03-07T07:00:00-10:00。

以下函数调整 dateTime 值 2002 年 3 月 7 日上午 10 点。*dateTime-value* 没有时区部分，而函数指定 *timezone-value* -PT10H。

```
fn:adjust-dateTime-to-timezone(xs:dateTime("2002-03-07T10:00:00"), $tz)
```

返回的 dateTime 为 2002-03-07T10:00:00-10:00。

以下函数调整 dateTime 值 2006 年 6 月 4 日上午 10 点（UTC-7 时区）。未指定 *timezone-value* 时，函数使用缺省时区 PT0H。

```
fn:adjust-dateTime-to-timezone(xs:dateTime("2006-06-04T10:00:00-07:00"))
```

返回的 dateTime 为 2006-06-04T17:00:00Z，dateTime 调整为 UTC。

以下函数除去 dateTime 值 2002 年 3 月 7 日上午 10 点（UTC-7 时区）的时区部分。*timezone-value* 值为空序列。

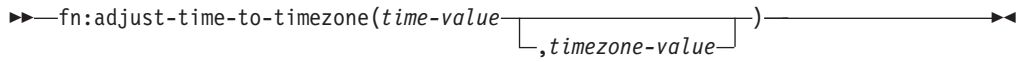
```
fn:adjust-dateTime-to-timezone(xs:dateTime("2002-03-07T10:00:00-07:00"), ())
```

返回的 dateTime 为 2002-03-07T10:00:00。

adjust-time-to-timezone 函数

fn:adjust-time-to-timezone 函数调整特定时区的 xs:time 值或除去该值的时区部分。

语法



time-value

要调整的时间值。

time-value 的类型为 xs:time，或者是空序列。

timezone-value

表示 *time-value* 要调整至的时区的持续时间。

timezone-value 可以是空序列或类型为 xdt:dayTimeDuration 的单值，范围在 -PT14H 到 PT14H 之间（包含两者）。该值可以是整数分钟值，并且不能有秒数部分。如果未指定 *timezone-value*，那么缺省值为 PT0H，表示 UTC。

返回的值

返回的值是类型为 xs:time 的值，或者是空序列，这取决于指定的参数。如果 *time-value* 不是空序列，那么返回的值类型为 xs:time。下表描述可能的返回值:

表 35. fn:adjust-time-to-timezone 的输入值和返回值的类型

date-value	*timezone-value*	返回的值
time-value，包含时区部分	显式值，或者未指定任何值（持续时间为 PT0H）	针对 *timezone-value* 表示的时区调整 *time-value*。返回的值包含 *timezone-value* 表示的时区部分。如果时区调整时跨过午夜，那么日期更改会被忽略。
time-value，包含时区部分	空序列	*time-value*，没有时区部分。
time-value，不包含时区部分	显式值，或者未指定任何值（持续时间为 PT0H）	*time-value*，带有时区部分。时区部分是由 *timezone-value* 表示的时区。未针对该时区调整时间部分。
time-value，不包含时区部分	空序列	*time-value*。
空序列	显式值、空序列或未指定任何值	空序列。

示例

在下列示例中，变量 $tz 为持续时间 -10 小时，定义为 xdt:dayTimeDuration("-PT10H")。

以下函数将时间值上午 10:00（UTC-7 时区），并且函数指定 *timezone-value* -PT10H。

```
fn:adjust-time-to-timezone(xs:time("10:00:00-07:00"), $tz)
```

返回的值为 7:00:00-10:00。时间调整为持续时间 -PT10H 表示的时区。

以下函数调整时间值下午 1:00。时间值没有时区部分。

```
fn:adjust-time-to-timezone(xs:time("13:00:00"), $tz)
```

返回的值为 13:00:00-10:00。时间包含持续时间 -PT10H 表示的时区部分。

以下函数调整时间值下午 10:00（UTC-7 时区）。该函数未指定 *timezone-value* 并使用缺省值 PT0H。

```
fn:adjust-time-to-timezone(xs:time("10:00:00-07:00"))
```

返回的值为 17:00:00Z，时间调整为 UTC。

以下函数除去时间值上午 8:00（UTC-7 时区）的时区部分。*timezone-value* 为空序列。

```
fn:adjust-time-to-timezone(xs:time("08:00:00-07:00"), ())
```

返回的值为 8:00:00。

以下示例比较两个时间。如果时区调整时跨过午夜，那么会导致日期更改。但是，fn:adjust-time-to-timezone 会忽略日期更改。

```
fn:adjust-time-to-timezone(xs:time("01:00:00+14:00"), $tz)
   eq xs:time("01:00:00-10:00")
```

返回的值为 true。

abs 函数

fn:abs 函数返回数字值的绝对值。

语法

numeric-value

原子值或空序列。

如果 *numeric-value* 是原子值，那么它具有下列其中一种类型:

- xs:float
- xs:double
- xs:decimal
- xs:integer
- 从上面列示的任一类型派生的类型
- xdt:untypedAtomic

如果 *numeric-value* 的数据类型为 xdt:untypedAtomic，那么它会转换为 xs:double 值。

返回的值

如果 *numeric-value* 并非空序列，那么返回的值是 *numeric-value* 的绝对值。

如果 *numeric-value* 是空序列，那么 fn:abs 会返回空序列。

返回值的数据类型取决于 *numeric-value* 的数据类型:

- 如果 *numeric-value* 的数据类型为 xs:float、xs:double、xs:decimal 或 xs:integer，那么返回的值的类型与 *numeric-value* 相同。
- 如果 *numeric-value* 的数据类型派生自 xs:float、xs:double、xs:decimal 或 xs:integer，那么返回值的数据类型为 *numeric-value* 的父代数据类型。
- 如果 *numeric-value* 的数据类型为 xdt:untypedAtomic，那么返回值的数据类型为 xs:double。

示例

以下函数返回 -10.5 的绝对值。

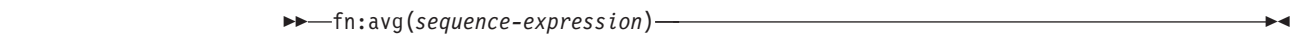
```
fn:abs(-10.5)
```

返回的值为 10.5。

avg 函数

fn:avg 函数返回序列中的平均值。

语法



sequence-expression

包含下列任一原子类型的项序列或空序列:

- xs:float
- xs:double
- xs:decimal
- xs:integer
- xdt:untypedAtomic
- xdt:dayTimeDuration
- xdt:yearMonthDuration
- 从上面列示的任一类型派生的类型

类型为 xdt:untypedAtomic 的输入项将转换为 xs:double。此次强制类型转换后，输入序列中的所有项必须可通过提升或子类型替换从而转换为公共类型。平均值是使用此公共类型计算的。例如，如果输入序列包含类型为 money（派生自 xs:decimal）和 stockprice（派生自 xs:float）的项，那么平均值将使用类型 xs:float 进行计算。

返回的值

如果 *sequence-expression* 并非空序列，那么返回的值是 *sequence-expression* 中各值的平均值。返回值的数据类型与 *sequence-expression* 中各项的数据类型相同，或者是 *sequence-expression* 中各项提升至的数据类型。

如果 *sequence-expression* 是空序列，那么返回空序列。

示例

以下函数返回序列 (5, 1.0E2, 40.5) 的平均值:

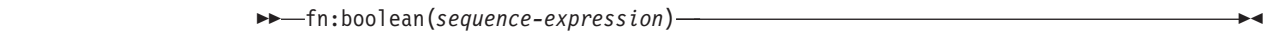
```
fn:avg((5, 1.0E2, 40.5))
```

这些值将提升至 xs:double 数据类型。该函数将返回 xs:double 值 4.85E1，它将以序列化方式表示为“48.5”。

boolean 函数

fn:boolean 函数返回序列的有效布尔值。

语法



sequence-expression
包含任何类型的项的任何序列或空序列。

返回的值

返回的有效布尔值（EBV）取决于 *sequence-expression* 的值:

表 36. 对 XQuery 中的特定值类型返回的 EBV

值的描述	返回的 EBV
空序列	false
第一项为节点的序列	true
类型为 xs:boolean 或派生自 xs:boolean 的单个值	false – 如果 xs:boolean 值为 false
	true – 如果 xs:boolean 值为 true
类型为 xs:string 或 xdt:untypedAtomic 或派生自其中一种类型的单个值	false – 如果值长度为零
	true – 如果值长度大于零
任意数字类型或派生自数字类型的单个值	false – 如果值为 NaN 或者其数值等于零
	true – 如果该值的数值不等于零
所有其他值	错误

注: 对于包含至少一个节点和至少一个原子值的序列，其有效布尔值在顺序不可预测的查询中是不确定的。

示例

自变量为单个数值的示例: 以下函数返回 0 的有效布尔值:

```
fn:boolean(0)
```

返回的值为 false。

自变量为包含多项的序列的示例: 以下函数返回 (<a/>, 0, <b/>) 的有效布尔值:

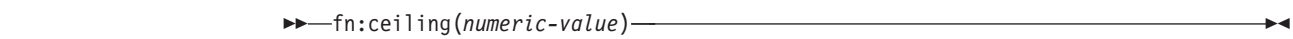
```
fn:boolean((<a/>, 0, <b/>))
```

返回的值为 true。

ceiling 函数

fn:ceiling 函数返回大于或等于特定数字值的最小整数。

语法



numeric-value

原子值或空序列。

如果 *numeric-value* 是原子值，那么它具有下列其中一种类型:

- xs:float
- xs:double
- xs:decimal
- xs:integer
- xdt:untypedAtomic
- 从上面列示的任一类型派生的类型

如果 *numeric-value* 的数据类型为 xdt:untypedAtomic，那么它会转换为 xs:double 值。

返回的值

如果 *numeric-value* 并非空序列，那么返回的值是大于或等于 *numeric-value* 的最小整数。返回值的数据类型取决于 *numeric-value* 的数据类型:

- 如果 *numeric-value* 的数据类型为 xs:float、xs:double、xs:decimal 或 xs:integer，那么返回的值的类型与 *numeric-value* 相同。
- 如果 *numeric-value* 的数据类型派生自 xs:float、xs:double、xs:decimal 或 xs:integer，那么返回值的数据类型为 *numeric-value* 的父代数据类型。

如果 *numeric-value* 是空序列，那么返回的值是空序列。

示例

使用正自变量的示例: 以下函数返回 0.5 的向上取整值:

```
fn:ceiling(0.5)
```

返回的值为 1。

使用负自变量的示例: 以下函数返回 (-1.2) 的向上取整值:

```
fn:ceiling(-1.2)
```

返回的值为 -1。

codepoints-to-string 函数

fn:codepoints-to-string 函数返回 Unicode 代码点序列的等价字符串。

语法

```
►►—fn:codepoints-to-string(codepoint-sequence)———————————————————►◄
```

codepoint-sequence
对应于 Unicode 代码点的整数序列或空序列。

返回的值

如果 *codepoint-sequence* 并非空序列，那么返回的值是一个字符串，由 *codepoint-sequence* 中各项的字符等价项并置而成。如果 *codepoint-sequence* 中的任何项并非有效 Unicode 代码点，那么会返回错误。

如果 *codepoint-sequence* 是空序列，那么返回的值为零长度字符串。

示例

以下函数返回 UTF-8 代码点序列 (88,81,117,101,114,121) 的字符等价项。

```
fn:codepoints-to-string((88,81,117,101,114,121))
```

返回的值为“XQuery”。

compare 函数

fn:compare 函数比较两个字符串。

语法

```
►►—fn:compare(string-1,string-2)———————————————————————————————►◄
```

string-1 , *string-2*
要比较的 xs:string 值。

返回的值

如果 *string-1* 和 *string-2* 并非空序列，那么会返回下列其中一个值:

-1 条件是 *string-1* 小于 *string-2*。

0 条件是 *string-1* 等于 *string-2*。

1 条件是 *string-1* 大于 *string-2*。

如果*string-1* 与 *string-2* 的长度相等（包括零长度），并且所有对应字符相等（根据缺省整理），那么两者相等。

如果 *string-1* 与 *string-2* 不相等，那么它们的关系（即，谁的值较高）是通过从字符串左端比较第一对不相等的字符来确定的。此比较是根据缺省整理进行的。

如果 *string-1* 比 *string-2* 长，并且 *string-2* 的所有字符等于 *string-1* 的前导字符，那么 *string-1* 大于 *string-2*。

如果 *string-1* 或 *string-2* 是空序列，那么返回空序列。

示例

以下函数使用缺省整理比较“ABC”与“ABD”。

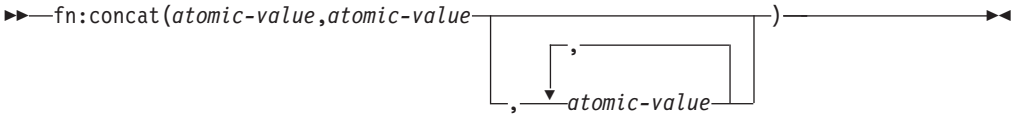
```
fn:compare('ABC', 'ABD')
```

“ABC”小于“ABD”。返回的值为 -1。

concat 函数

fn:concat 函数返回两个或更多原子值并置的字符串。

语法



atomic-value

原子值或空序列。如果自变量为空序列，那么该自变量会被视为零长度字符串。如果 *atomic-value* 并非 xs:string 值，那么它会在并置之前转换为 xs:string。

返回的值

如果所有 *atomic-value* 自变量都是空序列，那么返回的值为零长度字符串。否则返回的值为将 *atomic-value* 自变量转换为字符串时生成的 xs:string 值的并置。

示例

以下函数会将字符串“ABC”、“ABD”、空序列和“ABE”并置。

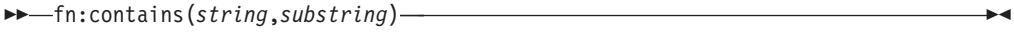
```
fn:concat('ABC', 'ABD', (), 'ABE')
```

返回的值为“ABCABDABE”。

contains 函数

fn:contains 函数确定字符串是否包含特定子串。使用缺省整理来匹配搜索字符串。

语法



string 用于搜索 *substring* 的字符串。

string 的数据类型为 xs:string，或者是空序列。如果 *string* 是空序列，那么 *string* 设置为零长度字符串。

substring
要在 *string* 中搜索的子串。

substring 的数据类型为 xs:string，或者是空序列。

对于长度的限制

substring 的长度不能超过 32000 个字节。

返回的值

如果满足下列任一条件，那么返回的值为 xs:boolean 值 true:

- *substring* 出现在 *string* 中的任何位置。
- *substring* 是空序列或零长度字符串。

否则，返回的值为 false。

示例

以下函数确定字符串“Test literal”是否包含字符串“lite”。

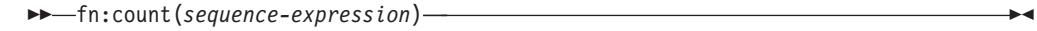
```
fn:contains('Test literal','lite')
```

返回的值为 true。

count 函数

fn:count 函数返回序列中的若干值。

语法



sequence-expression
包含任何类型的项的序列或空序列。

返回的值

如果 *sequence-expression* 并非空序列，那么返回 *sequence-expression* 中包含的值的数目。
如果 *sequence-expression* 是空序列，那么返回 0。

示例

以下函数返回序列 (5, 1.0E2, 40.5) 中包含的项数:

```
fn:count((5, 1.0E2, 40.5))
```

返回的值为 3。

current-date 函数

fn:current-date 函数返回 UTC 的隐式时区的当前日期。

语法

返回的值

返回的值为当前日期的 xs:date 值。

示例

以下函数返回当前日期。

```
fn:current-date()
```

如果在 2005 年 12 月 2 日调用此函数，那么返回的值为 2005-12-02Z。

current-dateTime 函数

fn:current-dateTime 函数返回 UTC 的隐式时区的当前日期和时间。

语法

返回的值

返回的值为当前日期和时间的 xs:dateTime 值。

示例

以下函数返回当前日期和时间。

```
fn:current-dateTime()
```

如果在多伦多（时区为 -PT5H）于 2005 年 12 月 2 日 6:25 调用此函数，那么返回的值为 2005-12-02T011:25:30.864001Z。

current-local-date 函数

db2-fn:current-local-date 函数返回本地时区的当前日期。

语法

返回的值

返回的值为当前日期的 xs:date 值。返回的值不包括时区部分。

例如，如果于格林威治标准时间（GMT）2009 年 12 月 2 日 3:00 调用此函数，且本地时区是东部标准时间（-PT5H），那么返回的值为 2009-12-01。

current-local-dateTime 函数

db2-fn:current-local-dateTime 函数返回本地时区的当前日期和时间。

语法

返回的值

返回的值为当前日期和时间的 xs:dateTime 值。返回的值不包括时区部分。

例如，如果在多伦多（时区为 -PT5H）于 2009 年 12 月 2 日 6:25 调用此函数，那么返回的值示例为 2009-12-02T06:25:30.864001。

current-local-time 函数

db2-fn:current-local-time 函数返回本地时区的当前时间。

语法

返回的值

返回的值为当前时间的 an xs:time 值。返回的值不包括时区部分。

例如，如果于格林威治标准时间（GMT）6:31 调用此函数，且本地时区是东部标准时间（-PT5H），那么示例返回的值为 01:31:35.519001。

current-time 函数

fn:current-time 函数返回 UTC 的隐式时区的当前时间。

语法

返回的值

返回的值为当前时间的 an xs:time 值。

示例

以下函数返回当前时间。

```
fn:current-time()
```

如果在全球标准时间 6:31 调用此函数，那么返回的值将为 06:31:35.519001Z。

data 函数

fn:data 函数在将输入序列中的任何节点替换为其类型值后返回输入序列。

语法

```
►►—fn:data(sequence-expression)———————————————————————————►◄
```

sequence-expression
任何序列，包括空序列。

返回的值

如果 *sequence-expression* 是空序列，那么返回的值是空序列。

如果 *sequence-expression* 是单个原子值，那么返回的值为 *sequence-expression*。

如果 *sequence-expression* 是单个节点，那么返回的值为 *sequence-expression* 的类型值。

如果 *sequence-expression* 是包含多项的序列，那么会根据 *sequence-expression* 中的各项返回原子值序列。*sequence-expression* 中的每个原子值保持不变。*sequence-expression* 中的每个节点被替换为其类型值，即包含零个或零个以上原子值的序列。

示例

以下函数返回包含序列 (<x xsi:type=″string″>ABC</x>,<y xsi:type=″decimal″>1.23</y>) 中的原子值的序列。

```
fn:data((<x xsi:type="string">ABC</x>,<y xsi:type="decimal">1.23</y>))
```

返回的值为 (″ABC″,1.23)。

dateTime 函数

fn:dateTime 函数根据 xs:date 值和 xs:time 值构造 xs:dateTime 值。

语法

```
►►—fn:dateTime(date-value,time-value)———————————————————————►◄
```

date-value
xs:date 值。

time-value
xs:time 值。

返回的值

返回的值是一个 xs:dateTime 值，其日期部分等于 *date-value*，时间部分等于 *time-value*。结果的时区是按如下方式计算的:

- 如果两个自变量都没有时区，那么结果也没有时区。
- 如果只有一个自变量有时区，或者两个自变量时区相同，那么结果使用此时区。
- 如果两个自变量的时区不同，那么会返回错误。

示例

以下函数根据 xs:date 值和 xs:time 值返回 xs:dateTime 值。

```
fn:dateTime((xs:date("2005-04-16")), (xs:time("12:30:59")))
```

返回的值为 xs:dateTime 值 2005-04-16T12:30:59。

day-from-date 函数

fn:day-from-date 函数返回 xs:date 值的日期部分。

语法

date-value

要从中抽取日期部分的日期值。

date-value 的类型为 xs:date，或者为空序列。

返回的值

如果 *date-value* 的类型为 xs:date，那么返回的值类型为 xs:integer，并且该值在 1 到 31 之间（包含两者）。该值是 *date-value* 的日期部分。

如果 *date-value* 是空序列，那么返回的值是空序列。

示例

以下函数返回日期值 2005 年 1 月 1 日的日期部分。

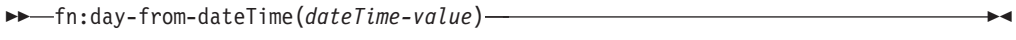
```
fn:day-from-date(xs:date("2005-06-01"))
```

返回的值为 1。

day-from-dateTime 函数

fn:day-from-dateTime 函数返回 xs:dateTime 值的日期部分。

语法



dateTime-value

要从中抽取日期部分的 dateTime 值。

dateTime-value 的类型为 xs:dateTime，或者是空序列。

返回的值

如果 *dateTime-value* 的类型为 xs:dateTime，那么返回的值类型为 xs:integer，并且该值在 1 到 31 之间（包含两者）。该值是 *dateTime-value* 的日期部分。

如果 *dateTime-value* 是空序列，那么返回的值是空序列。

示例

以下函数返回 dateTime 值 2005 年 1 月 31 日下午 8:00（UTC+4 时区）的日期部分。

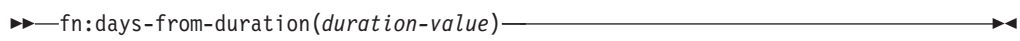
```
fn:day-from-dateTime(xs:dateTime("2005-01-31T20:00:00+04:00"))
```

返回的值为 31。

days-from-duration 函数

fn:days-from-duration 函数返回持续日期的天数部分。

语法



duration-value

要从中抽取天数部分的持续时间值。

duration-value 是空序列，或者是下列其中一个类型的值：xdt:dayTimeDuration、xs:duration 或 xdt:yearMonthDuration。

返回的值

返回值取决于 *duration-value* 的类型：

- 如果 *duration-value* 的类型为 xdt:dayTimeDuration 或 xs:duration，那么返回的值类型为 xs:integer，并且是转换为 xdt:dayTimeDuration 的 *duration-value* 的天数部分。如果 *duration-value* 为负值，那么返回的值为负值。
- 如果 *duration-value* 的类型为 xdt:yearMonthDuration，那么返回的值为 0。
- 如果 *duration-value* 是空序列，那么返回的值是空序列。

转换为 xdt:dayTimeDuration 的 *duration-value* 的天数部分是通过 (*S* idiv 86400) 计算的整数天数值。值 *S* 是转换为 xdt:dayTimeDuration 的 *duration-value* 的总秒数，用于除去年数和月数部分。

示例

此函数返回持续时间为 -10 天 0 小时的天数部分。

```
fn:days-from-duration(xdt:dayTimeDuration("-P10DT00H"))
```

返回的值为 -10。

此函数返回持续时间为 3 天 55 小时的天数部分。

```
fn:days-from-duration(xdt:dayTimeDuration("P3DT55H"))
```

返回的值为 5。计算持续时间的总天数时，55 小时转换为 2 天又 7 小时。持续时间等于 P5D7H，其天数部分为 5 天。

deep-equal 函数

fn:deep-equal 函数比较两个序列以确定它们是否符合深度相等的要求。

语法

sequence-1, *sequence-2*
要比较的序列。每个序列中的项可以是任何类型的原子值或节点。

返回的值

如果 *sequence-1* 和 *sequence-2* 深度相等，那么返回的值为 xs:boolean 值 true。否则，返回的值为 false。

如果 *sequence-1* 和 *sequence-2* 是空序列，那么它们的深度相等。

如果两个序列都不是空序列，并且同时符合下列两个条件，那么两个序列的深度相等:

- *sequence-1* 中的项数等于 *sequence-2* 中的项数。
- *sequence-1* 中的每一项（*item-1*）与 *sequence-2* 中的对应项（*item-2*）符合深度相等的条件。如果 *item-1* 和 *item-2* 符合下列任一条件，那么它们的深度相等:
  - *item-1* 和 *item-2* 都是原子值并且符合下列任一条件:
    - 表达式*item-1* eq *item-2* 返回 true
    - *item-1* 和 *item-2* 的类型都为 xs:float 或 xs:double 并且具有值 NaN。
  - *item-1* 和 *item-2* 是同类节点并且符合下表中的深度相等条件。

表 37. 序列中的节点深度相等

item-1 和 item-2 的节点种类	**深度相等的条件**
文档	item-1 的文本和元素子代序列与 item-2 的文本和元素子代序列的深度相等。
元素	必须符合下列所有条件: • *item-1* 和 *item-2* 名称相同，这表示它们的名称空间 URI 和局部名相匹配。名称空间前缀将被忽略。使用二进制比较来完成名称匹配。 • *item-1* 和 *item-2* 的属性数目相同，并且 *item-1* 的每个属性与 *item-2* 的某个属性的深度相等。 • 下列其中一个条件为真: – 两个节点都未经过验证或已使用允许混合内容（文本和子元素）的类型进行了验证，并且 *item-1* 的文本和元素子代序列与 *item-2* 的文本和元素子代序列深度相等。 – 两个节点都使用简单类型（如 xs:decimal）或具有简单内容的类型（如内容为 xs:decimal 的“temperature”类型）进行了验证，并且 *item-1* 的类型值与 *item-2* 的类型值深度相等。 – 两个节点都使用不允许任何内容（既不允许文本也不允许子元素）的类型进行了验证。 – 两个节点都使用仅允许子元素（不允许文本）的类型进行了验证，并且 *item-1* 的每个子元素与 *item-2* 的对应子元素深度相等。

表 37. 序列中的节点深度相等 (续)

item-1 和 item-2 的节点种类	**深度相等的条件**
属性	必须符合下列所有条件: • *item-1* 和 *item-2* 名称相同，这表示它们的名称空间 URI 和局部名相匹配。名称空间前缀将被忽略。使用二进制比较来完成名称匹配。 • *item-1* 的类型值与 *item-2* 的类型值深度相等。
文本	使用 eq 运算符作为字符串进行比较（通过缺省整理）时，内容属性值相等。
注释	使用 eq 运算符作为字符串进行比较（通过缺省整理）时，内容属性值相等。
处理指令	必须符合下列所有条件: • *item-1* 和 *item-2* 具有相同名称。 • 使用 eq 运算符作为字符串进行比较（通过缺省整理）时，内容属性值相等。

示例

以下函数会比较序列 (1,'ABC') 和 (1,'ABCD') 的深度是否相等。字符串比较使用缺省关联。

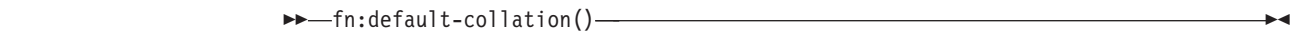
```
fn:deep-equal((1,'ABC'), (1,'ABCD'))
```

返回的值为 false。

default-collation 函数

fn:default-collation 函数返回表示表示为数据库定义的缺省整理的 URI。

语法



返回的值

返回的值的类型为 xs:anyURI，并且会指定数据库整理。

示例

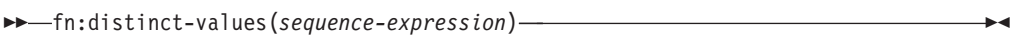
DB2 数据库是在将 CLDR181_LEN 指定为整理的情况下创建的。使用 fn:default-collation 函数查询此数据库时，返回以下值:

```
http://www.ibm.com/xmlns/prod/db2/sql/collations?name=CLDR181_LEN_AN_CX_EX_FX_HX_NX_S3
```

distinct-values 函数

fn:distinct-values 函数返回序列中的相异值。

语法



sequence-expression
原子值序列或空序列。

返回的值

如果 *sequence-expression* 并非空序列，那么返回的值为包含 *sequence-expression* 中的不同值的序列。如果 *value1* eq *value2* 为 false（使用缺省整理），那么表示 *value1* 和 *value2* 的值不同。如果未对两个值定义 eq 运算符，那么这两个值被视为不同。

在对值进行比较之前，类型为 xdt:untypedAtomic 的值将转换成类型为 xs:string 的值。

对于 xs:float 和 xs:double 值，如果 *sequence-expression* 包含多个 NaN 值，那么会返回单个 NaN 值。

对于 xs:dateTime、xs:date 或 xs:time 值，会在比较之前根据时差调整这些值。如果某个值没有时区，那么会使用隐式时区（UTC）。

如果 *sequence-expression* 是空序列，那么返回空序列。

如果使用 *eq* 运算符进行比较后显示输入序列中的两个值相等但类型不同，那么结果序列中会出现其中任意一个值（而不是两个值）。结果序列不会保留输入序列的顺序。

示例

在对序列中的节点进行原子化之后，以下函数会返回序列中的不同值:

```
fn:distinct-values((1, 'a', 1.0, 'A', <greeting>Hello</greeting>))
```

返回的值可能是 (1, 'a', 'A', 'Hello') 或 (1.0, 'A', 'a', 'Hello')。

empty 函数

fn:empty 函数指示自变量是否为空序列。

语法

项　　任何数据类型的表达式或空序列。

返回的值

如果 *item* 是空序列，那么返回的值为 true。否则，返回的值为 false。

示例

以下示例使用 empty 函数来确定变量 $seq 中的序列是否为空序列。

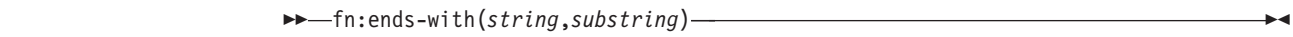
```
let $seq := (5, 10)
return fn:empty($seq)
```

返回的值为 false。

ends-with 函数

fn:ends-with 函数确定字符串是否以特定子串结束。使用缺省整理来匹配搜索字符串。

语法



string 用于搜索 *substring* 的字符串。

string 的数据类型为 xs:string，或者为空序列。如果 *string* 是空序列，那么 *string* 设置为零长度字符串。

substring

用于在 *string* 结尾进行搜索的子串。

substring 的数据类型为 xs:string，或者为空序列。

对于长度的限制

substring 的长度不能超过 32000 个字节。

返回的值

如果满足下列任一条件，那么返回的值为 xs:boolean 值 true:

- *substring* 出现在 *string* 结尾。
- *substring* 是空序列或零长度字符串。

否则，返回的值为 false。

示例

以下函数确定字符串“Test literal”是否以字符串“literal”结尾。

```
fn:ends-with('Test literal','literal')
```

返回的值为 true。

exactly-one 函数

如果自变量正好包含一项，fn:exactly-one 函数返回其自变量。

语法

sequence-expression

任何序列，包括空序列。

返回的值

如果 *sequence-expression* 正好包含一项，那么返回 *sequence-expression*。否则会返回错误。

示例

以下示例使用 exactly-one 函数来确定变量 $seq 中的序列是否正好包含一项。

```
let $seq := 5
return fn:exactly-one($seq)
```

返回的值为 5。

exists 函数

fn:exists 函数可以检查是否存在许多不同类型的项，例如，元素、属性、文本节点、静态值（例如，整数）或者 XML 文档。

如果指定为此函数的自变量的 XQuery 表达式 *sequence-expression* 产生了空结果（空序列），那么 fn:exists 将返回 **false**。如果此自变量返回的不是空序列，那么 fn:exists 将返回 **true**。

语法

sequence-expression
任何数据类型的序列或空序列。

返回的值

如果 *sequence-expression* 不是空序列，那么返回的值为 true。如果 *sequence-expression* 产生了空序列，那么返回的值为 false。

示例

以下示例使用 exists 函数来确定变量 $seq 中的序列是否为非空序列。

```
let $seq := (5, 10)
return fn:exists($seq)
```

返回的值为 true。

下一个示例将检查是否有一个元素 customer 具有子元素 phone。如果有，那么 fn:exists 函数将返回 true:

```
fn:exists($info/customer/phone)
```

如果有一个元素 customer，并且此元素具有 Cid 属性，那么以下示例将返回 true:

```
fn:exists($info/customer/@Cid)
```

下一个示例将检查 comment 元素是否具有文本节点。在此示例中，如果 comment 元素是一个空元素，那么它没有文本节点，因此 fn:exists 将返回 false。此外，如果根本就没有 comment 元素，那么 fn:exists 将返回 false:

```
fn:exists($info/customer/comment/text())
```

最后一个示例将检查 CUSTOMER 表的 XML 列 INFO 中是否有任何 XML 文档:

```
fn:exists( db2-fn:xmlcolumn("CUSTOMER.INFO") )
```

false 函数

fn:false 函数返回 xs:boolean 值 false。

语法

返回的值

返回的值为 xs:boolean 值 false。

示例

使用 false 函数以返回值 false。

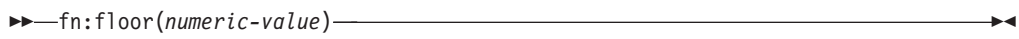
```
fn:false()
```

返回的值为 false。

floor 函数

fn:floor 函数返回小于或等于特定数字值的最大整数。

语法



numeric-value

原子值或空序列。

如果 *numeric-value* 是原子值，那么它具有下列其中一种类型:

- xs:float
- xs:double
- xs:decimal
- xs:integer
- xdt:untypedAtomic
- 从上面列示的任一类型派生的类型

如果 numeric-value 的数据类型为 xdt:untypedAtomic，那么它会转换为 xs:double 值。

返回的值

如果 *numeric-value* 并非空序列，那么返回的值是小于 *numeric-value* 的最大整数。返回值的数据类型取决于 *numeric-value* 的数据类型:

- 如果 *numeric-value* 的数据类型为 xs:float、xs:double、xs:decimal 或 xs:integer，那么返回的值的类型与 *numeric-value* 相同。
- 如果 *numeric-value* 的数据类型派生自 xs:float、xs:double、xs:decimal 或 xs:integer，那么返回值的数据类型为 *numeric-value* 的父代数据类型。

如果 *numeric-value* 是空序列，那么返回的值是空序列。

示例

使用正自变量的示例: 以下函数返回 0.5 的向下取整值:

```
fn:floor(0.5)
```

返回的值为 0。

使用负自变量的示例: 以下函数返回 (-1.2) 的向下取整值:

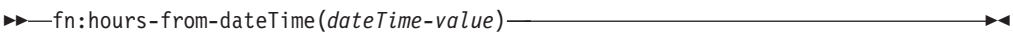
```
fn:floor(-1.2)
```

返回的值为 -2。

hours-from-dateTime 函数

fn:hours-from-dateTime 函数返回 xs:dateTime 值的小时数部分。

语法



dateTime-value

要从中抽取小时数部分的 dateTime 值。

dateTime-value 的类型为 xs:dateTime, 或者是空序列。

返回的值

如果 *dateTime-value* 的类型为 xs:dateTime, 那么返回的值类型为 xs:integer, 并且该值在 0 到 23 之间（包含两者）。该值是 *dateTime-value* 的小时数部分。

如果 *dateTime-value* 是空序列, 那么返回的值是空序列。

示例

以下函数返回 dateTime 值 2005 年 1 月 31 日下午 2:00（UTC-8 时区）的小时数部分。

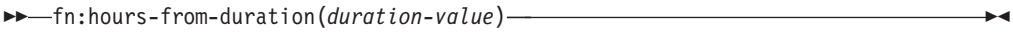
```
fn:hours-from-dateTime(xs:dateTime("2005-01-31T14:00:00-08:00"))
```

返回的值为 14。

hours-from-duration 函数

fn:hours-from-duration 函数返回持续时间值的小时数部分。

语法



duration-value

要从中抽取小时数部分的持续时间值。

duration-value 是空序列或者是下列其中一种类型的值：xdt:dayTimeDuration、xs:duration 和 xdt:yearMonthDuration。

返回的值

返回值取决于 *duration-value* 的类型：

- 如果 *duration-value* 的类型为 xdt:dayTimeDuration 或 xs:duration，那么返回的值类型为 xs:integer，并且是 -23 到 23 之间的值（包含两者）。该值是转换为 xdt:dayTimeDuration 的 *duration-value* 的小时数部分。如果 *duration-value* 为负值，那么该值为负值。
- 如果 *duration-value* 的类型为 xdt:yearMonthDuration，那么返回的值类型为 xs:integer 并且值为 0。
- 如果 *duration-value* 是空序列，那么返回的值是空序列。

转换为 xdt:dayTimeDuration 的 *duration-value* 的小时数部分是按 `((S mod 86400) idiv 3600)` 计算的整数小时数。*S* 是转换为 xdt:dayTimeDuration 的 *duration-value* 的总秒数以除去天数和月数部分。值 86400 是一天的秒数，3600 是一小时的秒数。

示例

以下函数返回持续时间 126 小时的小时数部分。

```
fn:hours-from-duration(xdt:dayTimeDuration("PT126H"))
```

返回的值为 6。计算持续时间的小时数时，126 小时转换为 5 天 6 小时。持续时间等于 P5DT6H，它的小时数部分为 6 小时。

hours-from-time 函数

fn:hours-from-time 函数返回 xs:time 值的小时数部分。

语法

time-value

要从中抽取小时数部分的时间值。

time-value 的类型为 xs:time，或者是空序列。

返回的值

如果 *time-value* 并非空序列，那么返回的值类型为 xs:integer，并且该值在 0 到 23 之间（包含两者）。该值是 *time-value* 的小时数部分。

如果 *time-value* 是空序列，那么返回的值是空序列。

示例

以下函数返回时间值上午 9:30（UTC-8 时区）的小时数部分。

```
fn:hours-from-time(xs:time("09:30:00-08:00"))
```

返回的值为 9。

implicit-timezone 函数

fn:implicit-timezone 函数返回隐式时区值 PT0S，该值的类型为 xs:dayTimeDuration。值 PT0S 指示 UTC 为隐式时区。

语法

返回的值

返回的值为 PT0S，这是由类型 xs:dayTimeDuration 表示的 UTC。

示例

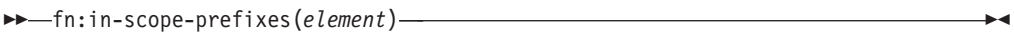
以下函数返回 xdt:dayTimeDuration(″PT0S″):

```
fn:implicit-timezone()
```

in-scope-prefixes 函数

fn:in-scope-prefixes 函数返回元素的所有名称空间作用域的前缀列表。

语法



element
要对其检索名称空间作用域前缀的元素。

返回的值

返回的值为 xs:NCName 值序列，这些值是 *element* 的所有名称空间作用域的前缀。如果缺省名称空间在 *element* 的作用域中，那么缺省名称空间前缀的序列项为零长度字符串。名称空间“xml”始终包括在元素的名称空间作用域中。

示例

以下查询返回元素 emp 的名称空间作用域的前缀（作为 NCName）序列。

```
declare namespace d="http://www.mycompany.com";
let $department := document {
    <comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
        <comp:emp id="31201" />
    </comp:dept> }
return fn:in-scope-prefixes($department/d:dept/d:emp)
```

返回的值为 (″xml″, ″comp″)，顺序不一定如此。

index-of 函数

fn:index-of 函数返回某项出现在序列中的位置。

语法

sequence-expression
原子值的任何序列或空序列。

search-value
要在 *sequence-expression* 中查找的值。

返回的值

返回的值是 xs:integer 值序列，使用 **eq** 运算符的规则进行比较（借助缺省整理）时，这些值表示 *sequence-expression* 中与 *search-value* 相匹配的项的位置。因为未对其类型定义 **eq** 运算符而未能进行比较的项将被视为与 *search-value* 不匹配，并且因此不会返回位置。序列中的第一项的位置为 1。

如果 *search-value* 在 *sequence-expression* 中没有匹配项，或者 *sequence-expression* 为空序列，那么函数会返回空序列。

示例

以下函数会返回“ABC”出现在序列中的位置。

```
fn:index-of(('ABC','DEF','ABC','123'), 'ABC')
```

返回的值为序列 (1,3)。

insert-before 函数

fn:insert-before 函数在一个序列中的特定位置之前插入另一序列。

语法

source-sequence
要在其中插入序列的序列。

source-sequence 是包含任意数据类型的项的序列或空序列。

insert-position
source-sequence 中之前要插入序列的位置。*insert-position* 的数据类型为 xs:integer。如果 *insert-position*<=0，那么 *insert-position* 会设置为 1。如果 *insert-position* 大于 *source-sequence* 中的项数，那么 *insert-position* 会设置为 *source-sequence* 中的项数加 1。

insert-sequence
要插入到 *source-sequence* 中的序列。

insert-sequence 是是包含任意数据类型的项的序列或空序列。

返回的值

如果 *source-sequence* 并非空序列:

- 如果 *insert-sequence* 并非空序列，那么返回的值为包含下列项并使用以下顺序的序列:
  - *source-sequence* 中的项在项 *insert-position* 之前
  - *insert-sequence* 中的项
  - *source-sequence* 中的项在项 *insert-position* 中
  - *source-sequence* 中的项在项 *insert-position* 之后
- 如果 *insert-sequence* 是空序列，那么返回的值为 *source-sequence*。

如果 *source-sequence* 是空序列:

- 如果 *insert-sequence* 并非空序列，那么返回的值为 *insert-sequence*。
- 如果 *insert-sequence* 是空序列，那么返回的值是空序列。

示例

以下函数返回因为在序列 (1,2,3,7) 中的位置 4 之前插入序列 (4,5,6) 而生成的序列:

```
fn:insert-before((1,2,3,7),4,(4,5,6))
```

返回的值为 (1,2,3,4,5,6,7)。

last 函数

fn:last 函数返回序列中正在处理的值的数目。

语法

返回的值

如果当前处理的序列并非空序列，那么返回的值为序列中的值的数目。如果当前处理的序列是空序列，那么返回的值是空序列。

示例

以下示例将该函数用作谓词表达式，以返回当前序列中的最后一项:

```
(<a/>, <b/>, <c/>)[fn:last()]
```

返回的值为 <c/>。

local-name 函数

fn:local-name 函数返回节点的局部名属性。

语法

node 要对其检索局部名的节点。如果未指定 *node*，那么会为当前上下文节点对 fn:local-name 求值。

返回的值

返回的值取决于是否指定了 *node* 及 *node* 的值:

- 如果未指定 *node*，那么会返回上下文节点的局部名。
- 如果 *node* 符合下列任一条件，那么会返回零长度字符串:
  - *node* 是空序列。
  - *node* 并非元素节点、属性节点或处理指令节点。
- 如果 *node* 符合下列任一条件，那么会返回错误:
  - *node* 未定义。
  - *node* 并非节点。
- 否则会返回 xs:string 值，该值包含 *node* 的扩展名称的局部名部分。

示例

以下函数返回节点 emp 的局部名。

```
declare namespace a="http://posample.org";
fn:local-name(<a:b/>)
```

返回的值为 b。

local-name-from-QName 函数

fn:local-name-from-QName 函数返回 xs:QName 值的局部。

语法

qualified-name

要从中检索局部的限定名称。

qualified-name 的数据类型为 xs:QName，或者是空序列。

返回的值

如果 *qualified-name* 并非空序列，那么返回的值为充当 *qualified-name* 的局部的 xs:NCName 值。如果 *qualified-name* 是空序列，那么返回空序列。

示例

以下函数返回限定名称的局部。

```
fn:local-name-from-QName(fn:QName("http://www.mycompany.com/", "ns:employee"))
```

返回的值为“employee”。

local-timezone 函数

db2-fn:local-timezone 函数返回本地系统的时区。

语法

返回的值

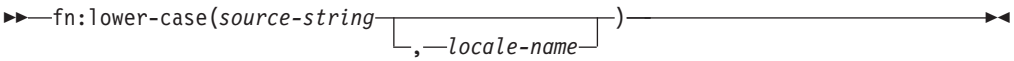
返回的值是 xdt:dayTimeDuration 值，它表示与全球标准时间（UTC）的本地时区偏移量。

例如，如果在东部标准时间的本地时区中调用此函数，那么返回的值是 -PT5H。

lower-case 函数

fn:lower-case 函数将字符串转换为小写。

语法



source-string

要转换为小写的字符串。

source-string 的类型为 xs:string，或者是空序列。

locale-name

包含要用于小写操作的语言环境的字符串。

locale-name 的类型为 xs:string，或者是空序列。如果 *locale-name* 并非空序列，那么 *locale-name* 的值是不区分大小写的，并且必须是有效语言环境或零长度字符串。

返回的值

如果 *source-string* 并非空序列，那么返回的值为 *source-string*，其中每个字符将转换为对应小写。如果 *locale-name* 未指定、为空序列或者是零长度字符串，那么会使用 Unicode 标准中定义的小写规则。否则会使用指定语言环境的小写规则。没有对应小写形式的每个字符将以其原始形式包含在返回的值中。

如果 *source-string* 是空序列，那么返回的值是零长度字符串。

示例

以下函数会将字符串“Wireless Router TB2561”转换为小写:

```
fn:lower-case("Wireless Router TB2561")
```

返回的值为“wireless router tb2561”。

以下函数指定土耳其语言环境 tr_TR 并转换字母“i”及数字字符引用 İ（拉丁大写 I 的字符引用，上面有点）。

```
fn:lower-case("I&#x130;", "tr_TR")
```

返回的值由两个字符组成: ı 表示的字符（拉丁小写无点 i）和字母“i”。对于土耳其语言环境，字母“I”将转换为 ı 表示的字符（拉丁小写无点 i），而 İ（拉丁大写 I，上面有点）将转换为“i”。

以下函数未指定语言环境，并使用 Unicode 标准中定义的规则将字母“I”转换为小写。

```
fn:lower-case("I")
```

返回的值是字母“i”。

matches 函数

fn:matches 函数确定字符串是否与特定模式相匹配。

语法

source-string
与模式进行比较的字符串。

source-string 是 xs:string 值或空序列。

pattern 要与 *source-string* 进行比较的正则表达式。正则表达式是用于在搜索模式中定义一个或一组字符串的字符、通配符和运算符集合。

pattern 是 xs:string 值。

flags 一个 xs:string 值，可包含下列用于控制 *pattern* 与 *source-string* 的匹配情况的任何值:

s 指示点（.）与任意字符相匹配。

如果未指定标志，那么点（.）与换行符（X'0A'）以外的任意字符相匹配。

m 指示插入标记（^）与行的开头（换行符之后的位置）相匹配，并且美元符号（$）与行结尾（换行符之前的位置）相匹配。

如果未指定 m 标志，那么插入标记（^）与字符串的开头相匹配，并且美元符号（$）与字符串结尾相匹配。

i 标识匹配是不区分大小写的。

如果未指定 i 标志，那么会执行区分大小写匹配。

x 指示 *pattern* 中的空格字符会被忽略。

如果未指定 x 标志，那么空格字符会用于匹配。

对于长度的限制

source-string 和 *pattern* 的长度不能超过 32000 个字节。

返回的值

如果 *source-string* 并非空序列，并且 *source-string* 与 *pattern* 相匹配，那么返回的值为 true。如果 *source-string* 与 *pattern* 不匹配，那么返回的值为 false。

如果 *pattern* 未包含字符串起始字符或行起始字符插入标记（^）或者字符串结束字符或行结束字符美元符号（$），并且 *source-string* 的任意子串与 *pattern* 相匹配，那么 *source-string* 与 *pattern* 相匹配。如果 *pattern* 包含字符串起始字符或行起始字符插入标记（^）或者字符串结束字符或行结束字符美元符号（$），那么仅当 *source-string* 与 *source-string* 开头或 *source-string* 中某行开头的 *pattern* 相匹配时，*source-string* 才与 *pattern* 相匹配。如果 *pattern* 包含字符串结束字符或行结束字符美元符号（$），那么仅当 *source-string* 与 *source-string* 结尾或 *source-string* 中某行结尾的 *pattern* 相匹配时，*source-string* 才与 *pattern* 相匹配。m 标志用于确定字符串开头或某行开头是否存在匹配。

如果 *source-string* 是空序列，那么返回的值为 false。

示例

使模式与字符串中的任意子串相匹配的示例：以下函数确定字符“ac”或“bd”是否出现在字符串“abbcacadbdcd”中的任意位置。

```
fn:matches("abbcacadbdcd","(ac)|(bd)")
```

返回的值为 true。

使模式与整个字符串相匹配的示例：以下函数确定字符“ac”或“bd”是否与字符串“bd”相匹配。

```
fn:matches("bd","^(ac)|(bd)$")
```

返回的值为 true。

匹配模式时忽略空格和首字母大写的示例：以下函数使用 i 和 x 标志，以在确定字符串“abc1234”是否与模式“ABC 1234”相匹配时忽略首字母大写和空格。

```
fn:matches("abc1234","ABC 1234", "ix")
```

返回的值为 true。

max 函数

fn:max 函数返回序列中的最大值。

语法

sequence-expression

包含下列任一原子类型的项序列或空序列:

- xs:float
- xs:double
- xs:decimal
- xs:integer
- xs:string
- xs:date
- xs:time
- xs:dateTime

- xdt:untypedAtomic
- xdt:dayTimeDuration
- xdt:yearMonthDuration
- 从上面列示的任一类型派生的类型

类型为 xdt:untypedAtomic 的输入项将转换为 xs:double。此次强制类型转换后，输入序列中的所有项必须可通过提升或子类型替换从而转换为支持 **ge** 运算符的公共类型。最大值是使用此公共类型计算的。例如，如果输入序列包含类型为 money（派生自 xs:decimal）和 stockprice（派生自 xs:float）的项，那么最大值将使用类型 xs:float 进行计算。

在比较日期、时间或日期时间值之前，会将它们调整为使用公共时区。没有显式时区部分的日期时间值将使用隐式时区，即 UTC。

使用缺省整理来比较字符串值。

返回的值

如果 *sequence-expression* 并非空序列，那么返回的值是 *sequence-expression* 中的值的最大值。返回值的数据类型与 *sequence-expression* 中各项的数据类型相同，或者是 *sequence-expression* 中各项提升至的公共数据类型。

如果 *sequence-expression* 是空序列，那么返回空序列。如果该序列包含值 NaN，那么会返回 NaN。

示例

以下函数返回序列 (500, 1.0E2, 40.5) 的最大值。

```
fn:max((500, 1.0E2, 40.5))
```

这些值将提升至 xs:double 数据类型。该函数将返回 xs:double 值 5.0E2，它将以序列化方式表示为“500”。

min 函数

fn:min 函数返回序列中的最小值。

语法

sequence-expression

包含下列任一原子类型的项序列或空序列:

- xs:float
- xs:double
- xs:decimal
- xs:integer
- xs:string
- xs:date
- xs:time
- xs:dateTime
- xdt:untypedAtomic

- xdt:dayTimeDuration
- xdt:yearMonthDuration
- 从上面列示的任一类型派生的类型

类型为 xdt:untypedAtomic 的输入项将转换为 xs:double。此次强制类型转换后，输入序列中的所有项必须可通过提升或子类型替换从而转换为支持 **le** 运算符的公共类型。最小值是使用此公共类型计算的。例如，如果输入序列包含类型为 money（派生自 xs:decimal）和 stockprice（派生自 xs:float）的项，那么最小值将使用类型 xs:float 进行计算。

在比较日期、时间或日期时间值之前，会将它们调整为使用公共时区。没有显式时区部分的日期时间值将使用隐式时区，即 UTC。

使用缺省整理来比较字符串值。

返回的值

如果 *sequence-expression* 并非空序列，那么返回的值是 *sequence-expression* 中的值的最小值。返回值的数据类型与 *sequence-expression* 中各项的数据类型相同，或者是 *sequence-expression* 中各项提升至的公共数据类型。

如果 *sequence-expression* 是空序列，那么返回空序列。如果该序列包含值 NaN，那么会返回 NaN。

示例

使用数字自变量的示例: 以下函数返回序列 (500, 1.0E2, 40.5) 的最小值:

```
fn:min((500, 1.0E2, 40.5))
```

这些值将提升至 xs:double 数据类型。该函数返回 xs:double 值 4.05E1，它将以序列化方式表示为“40.5”。

使用字符串自变量的示例: 以下函数使用缺省整理返回序列 (″x″, ″y″, ″Z″) 的最小值。假定缺省整理在排序时将小写字母字符排在大写字母字符之前。

```
fn:min(("x", "y", "Z"))
```

返回的值为“x”。

minutes-from-dateTime 函数

fn:minutes-from-dateTime 函数返回 xs:dateTime 值的分钟数部分。

语法

dateTime-value

要从中抽取分钟数部分的 dateTime 值。

dateTime-value 的类型为 xs:dateTime，或者是空序列。

返回的值

如果 *dateTime-value* 的类型为 xs:dateTime，那么返回的值类型为 xs:integer，并且该值在 0 到 59 之间（包含两者）。该值是 *dateTime-value* 的分钟数部分。

如果 *dateTime-value* 是空序列，那么返回的值是空序列。

示例

以下函数返回 dateTime 值 2005 年 1 月 23 日上午 9:42（UTC-8 时区）的分钟数部分。

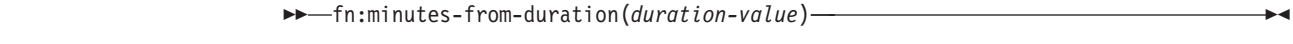
```
fn:minutes-from-dateTime(xs:dateTime("2005-01-23T09:42:00-08:00"))
```

返回的值为 42。

minutes-from-duration 函数

fn:minutes-from-duration 函数返回持续时间的分钟数部分。

语法



duration-value

要从中抽取分钟数部分的持续时间值。

duration-value 是空序列，或者是下列其中一个类型的值：xdt:dayTimeDuration、xs:duration 或 xdt:yearMonthDuration。

返回的值

返回值取决于 *duration-value* 的类型：

- 如果 *duration-value* 的类型为 xdt:dayTimeDuration 或 xs:duration，那么返回的值类型为 xs:integer，并且是 -59 到 59 之间的值（包含两者）。该值是转换为 xdt:dayTimeDuration 的 *duration-value* 的分钟数部分。如果 *duration-value* 为负值，那么该值为负值。
- 如果 *duration-value* 的类型为 xdt:yearMonthDuration，那么返回的值为 0。
- 如果 *duration-value* 是空序列，那么返回的值是空序列。

转换为 xdt:dayTimeDuration 的 *duration-value* 分钟数部分是通过 `((S mod 3600) idiv 60)` 计算的整数分钟数。值 *S* 是转换为 xdt:dayTimeDuration 的 *duration-value* 的总秒数，用于除去年数和月数部分。

示例

以下函数返回持续时间 2 天 16 小时 93 分钟的分钟数。

```
fn:minutes-from-duration(xdt:dayTimeDuration("P2DT16H93M"))
```

返回的值为 33。计算持续时间的总分钟数时，93 分钟将转换为 1 小时 33 分钟。持续时间等于 P2DT17H33M，它的分钟数部分为 33 分钟。

minutes-from-time 函数

fn:minutes-from-time 函数返回 xs:time 值的分钟数部分。

语法

time-value

要从中抽取分钟数部分的时间值。

time-value 的类型为 xs:time，或者是空序列。

返回的值

如果 *time-value* 的类型为 xs:time，那么返回的值类型为 xs:integer，并且该值在 0 到 59 之间（包含两者）。该值是 *time-value* 的分钟数部分。

如果 *time-value* 是空序列，那么返回的值是空序列。

示例

以下函数返回时间值上午 8:59（UTC-8 时区）的分钟数部分。

```
fn:minutes-from-time(xs:time("08:59:00-08:00"))
```

返回的值为 59。

month-from-date 函数

fn:month-from-date 函数返回 xs:date 值的月份部分。

语法

date-value

要从中抽取月份部分的日期值。

date-value 的类型为 xs:date，或者为空序列。

返回的值

如果 *date-value* 的类型为 xs:date，那么返回的值类型为 xs:integer，并且该值在 1 到 12 之间（包含两者）。该值是 *date-value* 的月份部分。

如果 *date-value* 是空序列，那么返回的值是空序列。

示例

以下函数返回日期值 2005 年 12 月 1 日的月份部分。

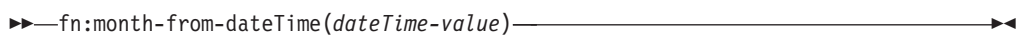
```
fn:month-from-date(xs:date("2005-12-01"))
```

返回的值为 12。

month-from-dateTime 函数

fn:month-from-dateTime 函数返回 xs:dateTime 值的月份部分。

语法



dateTime-value

要从中抽取月份部分的 dateTime 值。

dateTime-value 的类型为 xs:dateTime，或者是空序列。

返回的值

如果 *dateTime-value* 的类型为 xs:dateTime，那么返回的值类型为 xs:integer，并且该值在 1 到 12 之间（包含两者）。该值是 *dateTime-value* 的月份部分。

如果 *dateTime-value* 是空序列，那么返回的值是空序列。

示例

以下函数返回 dateTime 值 2005 年 10 月 31 日上午 8:15（UTC-8 时区）的月份部分。

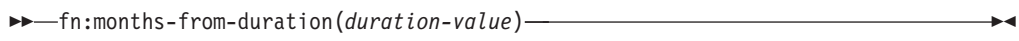
```
fn:month-from-dateTime(xs:dateTime("2005-10-31T08:15:00-08:00"))
```

返回的值为 10。

months-from-duration 函数

fn:months-from-duration 函数返回持续时间值的月数部分。

语法



duration-value

要从中抽取月数部分的持续时间值。

duration-value 是空序列，或者是下列其中一个类型的值：xdt:dayTimeDuration、xs:duration 或 xdt:yearMonthDuration。

返回的值

返回值取决于 *duration-value* 的类型：

- 如果 *duration-value* 的类型为 xs:duration 或 xdt:yearMonthDuration，那么返回的值类型为 xs:integer，并且是 -11 到 11 之间的值（包含两者）。该值是转换为 xdt:yearMonthDuration 的 *duration-value* 的月数部分。如果 *duration-value* 为负值，那么该值为负值。
- 如果 *duration-value* 的类型为 xdt:dayTimeDuration，那么返回的值为 0。
- 如果 *duration-value* 是空序列，那么返回的值是空序列。

转换为 xdt:yearMonthDuration 的 *duration-value* 月数部分是 *duration-value* 总月数除以 12 确定的余数确定的整数月数值。

示例

以下函数返回持续时间 20 年零 5 个月的月数部分。

```
fn:months-from-duration(xs:duration("P20Y5M"))
```

返回的值为 5。

以下函数返回 yearMonthDuration -9 年 -13 个月的月数部分。

```
fn:months-from-duration(xdt:yearMonthDuration("-P9Y13M"))
```

返回的值为 -1。计算持续时间的总月数时，-13 个月将转换为 -1 年 -1 个月。持续时间等于 -P10Y1M，其月数部分为 -1 个月。

以下函数返回持续时间 14 年零 11 个月 40 天又 13 小时的月数部分。

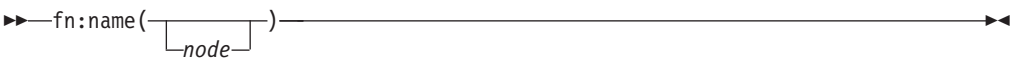
```
xquery fn:months-from-duration(xs:duration("P14Y11M40DT13H"))
```

返回的值为 11。

name 函数

fn:name 函数返回节点名的前缀和局部名部分。

语法



node 要对其检索名称的节点的限定名称。如果未指定 *node*，那么会为当前上下文节点对 fn:name 求值。

返回的值

返回的值取决于 *node* 的值:

- 如果 *node* 符合下列任一条件，那么会返回零长度字符串:
  - *node* 是空序列。
  - *node* 并非元素节点、属性节点或处理指令节点。
- 如果 *node* 符合下列任一条件，那么会返回错误:
  - *node* 未定义。
  - *node* 并非节点。
- 否则会返回 xs:string 值，该值包含 *node* 的前缀（如果存在）和局部名。

示例

以下查询返回值“comp:emp”:

```
declare namespace d="http://www.mycompany.com";
let $department := document {
    <comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
        <comp:emp id="31201" />
    </comp:dept> }
return fn:name($department/d:dept/d:emp)
```

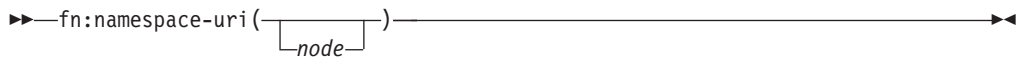
以下查询还会返回值“comp:emp”:

```
declare namespace d="http://www.mycompany.com";
let $department := document {
    <comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
        <comp:emp id="31201" />
    </comp:dept> }
return $department/d:dept/d:emp/fn:name()
```

namespace-uri 函数

fn:namespace-uri 函数返回节点的限定名称的名称空间 URI。

语法



node 要对其检索名称空间 URI 的节点的限定名称。如果未指定 *node*，那么会为当前上下文节点对 fn:namespace-uri 求值。

返回的值

返回的值取决于 *node* 的值:

- 如果 *node* 符合下列任一条件，那么会返回零长度字符串:
  - *node* 是空序列。
  - *node* 并非元素节点或属性节点。
  - *node* 是元素节点或属性节点，但 *node* 的扩展限定名称不在名称空间中。
- 如果 *node* 符合下列任一条件，那么会返回错误:
  - *node* 未定义。
  - *node* 并非节点。
- 否则会返回 xs:string 值，该值包含 *node* 的扩展名称的名称空间 URI。

示例

以下查询返回值“http://www.mycompany.com”:

```
declare namespace d="http://www.mycompany.com";
let $department := document {
    <comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
        <comp:emp id="31201" />
    </comp:dept> }
return fn:namespace-uri($department/d:dept/d:emp)
```

以下查询还会返回值“http://www.mycompany.com”:

```
declare namespace d="http://www.mycompany.com";
let $department := document {
    <comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
```

```
        <comp:emp id="31201" />
    </comp:dept> }
return $department/d:dept/d:emp/fn:namespace-uri()
```

namespace-uri-for-prefix 函数

fn:namespace-uri-for-prefix 函数返回与元素名称空间作用域中的前缀相关联的名称空间 URI。

语法

prefix 要对其返回名称空间的前缀。

prefix 的数据类型为 xs:string 并且长度可能为零，或者是空序列。

element

其名称空间作用域绑定至 *prefix* 的元素。

返回的值

返回的值取决于 *prefix* 的值:

- 如果 *element* 的名称空间作用域的前缀值与 *prefix* 的值相匹配，那么会返回该名称空间的名称空间 URI。
- 如果 *element* 没有其前缀值与 *prefix* 的值相匹配的名称空间作用域，那么会返回空序列。
- 如果 *prefix* 是零长度字符串或空序列，那么会返回缺省名称空间的名称空间 URI。

示例

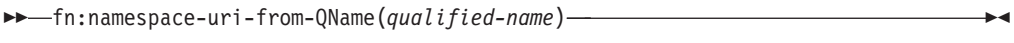
以下查询返回值“http://www.mycompany.com”:

```
declare namespace d="http://www.mycompany.com";
let $department := document {
    <comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
        <comp:emp id="31201" />
    </comp:dept> }
return fn:namespace-uri-for-prefix("comp", $department/d:dept/d:emp)
```

namespace-uri-from-QName 函数

fn:namespace-uri-from-QName 函数返回 xs:QName 值的名称空间 URI 部分。

语法



qualified-name

要从中检索名称空间 URI 部分的限定名称。

qualified-name 的数据类型为 xs:QName，或者是空序列。

返回的值

如果 *qualified-name* 并非空序列，那么返回的值为充当 *qualified-name* 的名称空间 URI 部分的 xs:string 值。如果 *qualified-name* 不在名称空间中，那么会返回零长度字符串。如果 *qualified-name* 是空序列，那么返回空序列。

示例

此函数返回字符串值“http://www.mycompany.com”:

```
fn:namespace-uri-from-QName(fn:QName("http://www.mycompany.com", "comp:employee"))
```

node-name 函数

fn:node-name 函数返回节点的扩展 QName。

语法

node 要对其检索扩展名称的节点。

返回的值

返回的值是包含 *node* 的扩展 QName 的 xs:QName 值。如果 *node* 是空序列，那么会返回空序列。

示例

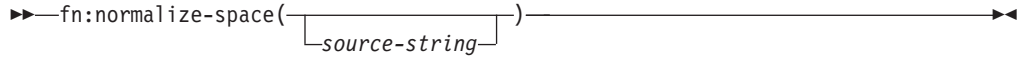
以下查询返回对应于 URI http://www.mycompany.com 和词汇 QName comp:emp 的扩展 QName:

```
declare namespace d="http://www.mycompany.com";
let $department := document {
    <comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
        <comp:emp id="31201" />
    </comp:dept> }
return fn:node-name($department/d:dept/d:emp)
```

normalize-space 函数

fn:normalize-space 函数除去字符串中的前导和尾部空格字符，并将内部的每个空格字符序列替换为单个空白字符。

语法



source-string

要规范化其中的空格的字符串。

source-string 是 xs:string 值或空序列。

如果未指定 *source-string*，那么 fn:normalize-space 的自变量是当前上下文项，它将通过使用 fn:string 函数转换为 xs:string 值。

返回的值

返回的值是对 *source-string* 执行下列操作时产生的 xs:string 值:

- 前导空格和结尾空格将被除去。
- 一个或多个相邻空格字符的每个内部序列将替换为单空格（X'20'）字符。

空格字符包括空格（X'20'）、制表符（X'09'）、换行符（X'0A'）和回车符（X'0D'）。

如果 *source-string* 是空序列，那么会返回零长度字符串。

示例

以下函数会从字符串"a b c d"中除去额外的空格字符。

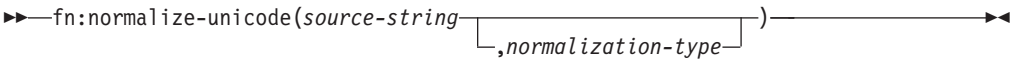
```
fn:normalize-space(" a b c d " )
```

返回的值为"a b c d"。

normalize-unicode 函数

fn:normalize-unicode 函数对字符串执行 Unicode 规范化。

语法



source-string

要对其执行 Unicode 规范化的值。

source-string 是 xs:string 值或空序列。

normalization-type

指示要执行的 Unicode 规范化的类型的 xs:string 值。可能的值包括:

NFC Unicode 规范化格式 C。如果未指定 *normalization-type*，那么会执行 Unicode 规范化。

NFD Unicode 规范化格式 D。

NFKC Unicode 规范化格式 KC。

NFKD Unicode 规范化格式 KD。

如果指定了零长度字符串，那么不会执行规范化。

返回的值

如果 *source-string* 并非空序列，那么返回的值是对 *source-string* 执行 *normalization-type* 指定的 Unicode 规范化时生成的 xs:string 值。如果未指定 *normalization-type*，那么会对 *source-string* 执行 Unicode 规范化格式 C（NFC）。*Character Model for the World Wide Web 1.0* 中对 Unicode 规范化作了描述。

如果 *source-string* 是空序列，那么会返回零长度字符串。

示例

以下函数对字符串“ṃ”（下面带点的拉丁小写字母 m）执行 Unicode 规范化格式 C:

```
fn:normalize-unicode("&#x6d;&#x323;","NFC")
```

返回的值是由数字字符引用 &x1e43;（下面带点的拉丁小写字母 m）表示的 UTF-8 字符。

以下示例将规范化 Unicode 转换为十进制代码点:

```
fn:string-to-codepoints(fn:normalize-unicode("&#x6d;&#x323;", "NFC"))
```

返回的值为 7747。

not 函数

如果序列的有效布尔值为 true，那么 fn:not 函数返回 false，如果序列的有效布尔值为 false，那么该函数返回 true。

语法

sequence-expression
包含任何类型的项的任何序列或空序列。

返回的值

如果 *sequence-expression* 并非空序列，并且序列的有效布尔值为 false，那么返回的值为 true。如果序列的有效布尔值为 true，那么返回的值为 false。

如果 *sequence-expression* 是空序列，那么返回的值为 true。

示例

以下函数返回 false，原因是节点的有效布尔值为 true。

```
fn:not(<employee />)
```

number 函数

fn:number 函数将值转换为 xs:double 数据类型。

语法

atomic-value
原子值或空序列。如果未指定 *atomic-value*，那么会为当前上下文项对 fn:number 求值。

返回的值

如果 *atomic-value* 并非空序列，那么返回的值是通过将 *atomic-value* 的强制类型转换为 xs:double 生成的。如果 *atomic-value* 不能转换为 xs:double 数据类型，那么会返回 NaN。

如果 *numeric-value* 是空序列，那么会返回 NaN。

示例

将 xs:decimal 值转换为 xs:double 的示例: 以下函数会将 xs:decimal 值 2.75 转换为 xs:double。

```
fn:number(2.75)
```

返回的值为 2.75E0。

将 xs:boolean 的值转换为 xs:double 的示例: 以下函数会将布尔值 false() 转换为 xs:double。

```
fn:number(false())
```

返回的值为 0.0E0。

one-or-more 函数

如果自变量包含一项或多项，那么 fn:one-or-more 函数返回其自变量。

语法

sequence-expression
任何序列，包括空序列。

返回的值

如果 *sequence-expression* 包含一项或多项，那么会返回 *sequence-expression*。否则会返回错误。

示例

以下示例使用 fn:one-or-more 函数来确定变量 $seq 中的序列是否包含一项或多项。

```
let $seq := (5,10)
return fn:one-or-more($seq)
```

返回 (5,10)。

position 函数

fn:position 函数返回序列中正在处理的上下文项的位置。

语法

返回的值

返回的值是一个 xs:integer 值，该值指示序列中当前处理的上下文项的位置。如果未定义上下文项，那么会返回错误。仅当包含上下文项的序列具有确定顺序时，position 函数才会返回确定结果。通常会在谓词中使用 position 函数。

示例

在以下表达式中，会对包含 10 项的序列中的每项调用 position 函数。对于每一项，position 函数都会返回该项在序列中的位置。谓词 `position() eq 5` 仅对序列中的第 5 项显示为 true。

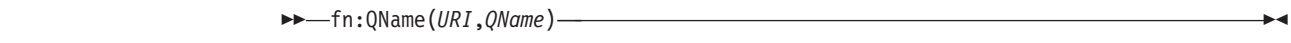
```
(11 to 20)[position() eq 5]
```

表达式返回的值为 15。

QName 函数

fn:QName 函数根据名称空间 URI 和包含词汇 QName（带有可选前缀）的字符串构建扩展名。

语法



URI 扩展名称的名称空间部分。

URI 的数据类型为 xs:string，或者为空字符串或序列。

QName

xs:QName 数据类型的正确词法格式的值。

QName 的数据类型为 xs:string。

返回的值

返回的值是充当扩展名称的 xs:QName 值，该名称具有由 *URI* 指定的名称空间 URI 以及由 *QName* 指定的前缀和局部名。

fn:QName 函数会使 *QName* 的名称空间前缀与 *URI* 的值相关联。如果 *QName* 具有名称空间前缀，那么 *URI* 不能是零长度字符串或空序列。如果 *QName* 只有局部名并且没有前缀，那么 *URI* 可以是零长度字符串或空序列。

示例

已对以下函数指定名称空间 URI 及包含词汇 QName 的字符串，并且此函数会返回类型为 xs:QName 的值。

```
fn:QName("http://www.mycompany.com", "comp:employee")
```

返回的值为 xs:QName 值，其名称空间 URI 为“http://www.mycompany.com”，前缀为“comp”，局部名为“employee”。

remove 函数

fn:remove 函数除去序列中的一项。

语法

source-sequence

从中除去一项的序列。

source-sequence 是包含任意数据类型的项的序列或空序列。

remove-position

要除去的项在 *source-sequence* 中的位置。*remove-position* 的数据类型为 xs:integer。

返回的值

如果 *source-sequence* 并非空序列:

- 如果 *remove-position* 小于 1 或大于 *source-sequence* 的长度，那么返回的值为 *source-sequence*。
- 如果 *remove-position* 大于或等于 1 并且小于或等于 *source-sequence* 的长度，那么返回的值是包含下列项并使用以下顺序的序列:
  - *source-sequence* 中的项在项 *remove-position* 之前
  - *source-sequence* 中的项在项 *remove-position* 之后
- 如果 *source-sequence* 是空序列，那么返回的值是空序列。

示例

以下函数返回因为除去序列 (1,2,4,7) 中位置 3 处的项而生成的序列:

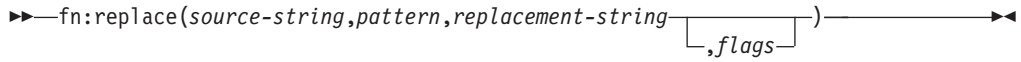
```
fn:remove((1,2,4,7),3)
```

返回的值为 (1,2,7)。

replace 函数

fn:replace 函数将字符串中的每组字符与特定模式进行比较，然后将与该模式匹配的字符替换为另一组字符。

语法



source-string
包含要替换的字符的字符串。

source-string 是 xs:string 值或空序列。

pattern 要与 *source-string* 进行比较的正则表达式。正则表达式是用于在搜索模式中定义一个或一组字符串的字符、通配符和运算符集合。

pattern 是 xs:string 值。

replacement-string
一个字符串，包含用于替换与 *source-string* 中的 *pattern* 相匹配的字符。

replacement-string 是 xs:string 值。

replacement-string 可包含变量 $0 到 $9。$0 表示 *pattern* 中的整个字符串。变量 $1 到 $9 表示 *pattern* 中 9 个可能带括号子表达式中的一个。（$1 表示第一个子表达式，$2 表示第二个子表达式，以此类推。）

要在 *replacement-string* 中使用文字美元符号“$”，请使用字符串“\$”。要在 *replacement-string* 中使用文字反斜杠（\），请使用字符串“\\”。

flags 一个 xs:string 值，可包含下列用于控制 *pattern* 与 *source-string* 的匹配情况的任何值:

s 表示点（.）可替换任何字符。

如果未指定标志，那么点（.）替换换行符（X'0A'）以外的任意字符。

m 表示插入标记（^）将替换行的开头（换行符之后的位置），而美元符号（$）替换行的结尾（换行符之前的位置）。

如果未指定 m 标志，那么插入标记（^）将替换字符串的开头，并且美元符号（$）会替换字符串的结尾。

i 标识匹配是不区分大小写的。

如果未指定 i 标志，那么会执行区分大小写匹配。

x 指示 *pattern* 中的空格字符会被忽略。

如果未指定 x 标志，那么空格字符会用于匹配。

对于长度的限制

source-string、*pattern* 和 *replacement-string* 的长度不能超过 32000 个字节。

返回的值

如果 *source-string* 并非空序列，那么返回的值是对 *source-string* 执行下列操作时生成的字符串:

- 将搜索 *source-string* 以查找与 *pattern* 相匹配的字符。如果 *pattern* 包含两组或更多备用字符集，那么与 *source-string* 中字符相匹配的 *pattern* 中的第一组字符被视为匹配模式。

- 与 *pattern* 匹配的 *source-string* 中的每组字符将替换为 *replacement-string*。如果 *replacement-string* 包含变量 $0 到 $9 中的任何一个，那么与对应该变量的 *pattern* 中子表达式相匹配的 *source-string* 子串将在 *replacement-string* 中替换该变量。然后经过修改的 *replacement-string* 将插入到 *source-string* 中。如果因为变量数超过子表达式数或者子表达式在 *source-string* 中没有匹配而导致变量在 *pattern* 中没有对应子表达式，那么零长度字符串会在 *replacement-string* 中替换该变量。

如果在 *source-string* 中找不到 *pattern*，那么会返回错误。

如果 *source-string* 是空序列，那么会返回零长度字符串。

示例

将一个子串替换为另一子串的示例：以下函数会将字符串“abbcacadbdcd”中的所有“a”的实例替换为“ba”。

```
fn:replace("abbcacadbdcd","a","ba")
```

返回的值为“babbcbacbadbdcd”。

将使用替换字符串的子串替换为变量的示例：以下函数在“abbcacadbdcd”中将“a”和跟在其后的字符替换为“a”之后的字符的两个实例。

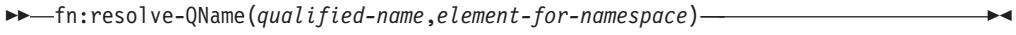
```
fn:replace("abbcacadbdcd","a(.)","$1$1")
```

返回的值为“bbbcccddbdcd”。

resolve-QName 函数

通过使用元素的名称空间作用域将名称空间前缀解析为名称空间 URI，fn:resolve-QName 函数将包含词汇 QName 的字符串转换为扩展 QName。

语法



qualified-name

使用限定名称格式的字符串。

qualified-name 的数据类型为 xs:string，或者是空序列。

element-for-namespace

为 *qualified-name* 提供名称空间作用域的元素。

element-for-namespace 是元素节点。

返回的值

如果 *qualified-name* 并非空序列，那么返回的值是按如下方式构造的扩展名称：

- 扩展 QName 的前缀和局部名取自 *qualified-name*。
- 如果 *qualified-name* 有前缀，并且该前缀与 *element-for-namespace* 的名称空间作用域中的前缀相匹配，那么此前缀绑定至的名称空间 URI 是返回值的名称空间 URI。

- 如果 *qualified-name* 没有前缀，并且缺省值名称空间 URI 是在 *element-for-namespace* 的名称空间作用域中定义的，那么此缺省名称空间 URI 是返回值的名称空间 URI。
- 如果 *qualified-name* 没有前缀，并且未在 *element-for-namespace* 的名称空间作用域中定义任何缺省值名称空间 URI，那么返回的值没有任何名称空间 URI。
- 如果 *qualified-name* 的前缀与 *element-for-namespace* 的名称空间作用域中的名称空间前缀不匹配，或者 *qualified-name* 并未使用有效限定名称格式，那么会返回错误。

如果 *qualified-name* 是空序列，那么返回空序列。

示例

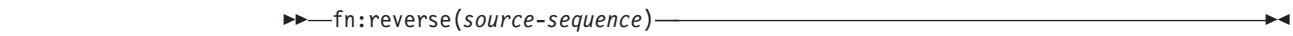
以下查询返回对应于 URI http://www.mycompany.com 和词汇 QName comp:dept 的扩展 QName:

```
declare namespace d="http://www.mycompany.com";
let $department := document {
<comp:dept xmlns:comp="http://www.mycompany.com" id="A07">
  <comp:emp id="31201" />
</comp:dept> }
return fn:resolve-QName("comp:dept", $department/d:dept/d:emp)
```

reverse 函数

fn:reverse 函数使序列中的项排序反向。

语法



source-sequence
要反向的序列。

source-sequence 是包含任意数据类型的项的序列或空序列。

返回的值

如果 *source-sequence* 并非空序列，那么返回的值是以逆向顺序包含 *source-sequence* 中各项的序列。

如果 *source-sequence* 是空序列，那么会返回空序列。

示例

以下函数以逆向顺序返回序列 (1,2,3,7) 中的各项:

```
fn:reverse((1,2,3,7))
```

返回的值为 (7,3,2,1)。

root 函数

fn:root 函数返回节点所属的树的根节点。

语法

node 节点或空序列。*node* 的缺省值是上下文节点。

返回的值

如果 *node* 并非空序列，那么返回的值是 *node* 所属的树的根节点。如果 *node* 是树的根节点，那么返回的值为 *node*。

如果 *node* 是空序列，那么返回的值是空序列。

示例

假定某些 XQuery 变量进行了如下定义:

```
let $f := <first>Laura</first>
let $e := <emp> {$f} <last>Brown</last> </emp>
let $doc := document {<emps>{$e}</emps>}
```

返回元素的根节点的示例: 以下函数返回元素 last 的根节点:

```
fn:root($e/last)
```

返回的值为 <emp><first>Laura</first><last>Brown</last></emp>。

返回文档的根节点的示例: 以下函数返回绑定至变量 $doc 的文档的根节点:

```
fn:root($doc)
```

返回的值是文档节点。

round 函数

fn:round 函数返回最接近特定数字值的整数。

语法

numeric-value

原子值或空序列。

如果 *numeric-value* 是原子值，那么它具有下列其中一种类型:

- xs:float
- xs:double
- xs:decimal
- xs:integer
- xdt:untypedAtomic
- 从上面列示的任一类型派生的类型

如果 *numeric-value* 的数据类型为 xdt:untypedAtomic，那么它会转换为 xs:double 值。

返回的值

如果 *numeric-value* 并非空序列，那么返回的值是最接近 *numeric-value* 的整数。即，fn:round(*numeric-value*) 等价于 fn:floor(*numeric-value*+0.5)。返回值的数据类型取决于 *numeric-value* 的数据类型:

- 如果 *numeric-value* 的数据类型为 xs:float、xs:double、xs:decimal 或 xs:integer，那么返回的值的类型与 *numeric-value* 相同。
- 如果 *numeric-value* 的数据类型派生自 xs:float、xs:double、xs:decimal 或 xs:integer，那么返回值的数据类型为 *numeric-value* 的父代数据类型。

如果 *numeric-value* 是空序列，那么返回的值是空序列。

示例

使用正自变量的示例: 以下函数返回 0.5 的舍入值:

```
fn:round(0.5)
```

返回的值为 1。

使用负自变量的示例: 以下函数返回 (-1.5) 的舍入值:

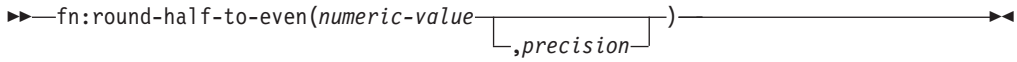
```
fn:round(-1.5)
```

返回的值为 -1。

round-half-to-even 函数

fn:round-half-to-even 函数返回最接近特定数字值并具有指定精度的数字值。

语法



numeric-value

原子值或空序列。

如果 *numeric-value* 是原子值，那么它具有下列其中一种类型:

- xs:float
- xs:double
- xs:decimal
- xs:integer
- xdt:untypedAtomic
- 从上面列示的任一类型派生的类型

如果 *numeric-value* 的数据类型为 xdt:untypedAtomic，那么它会转换为 xs:double 值。

precision

numeric-value 要舍入的小数点右边的位数。*precision* 是 xs:integer 值。*precision* 的缺省值为 0。

返回的值

如果 *numeric-value* 并非空序列，并且 *precision* 为 0 或未指定，那么返回的值为最接近 *numeric-value* 的整数。如果 *numeric-value* 与两个整数的接近程度相等，那么返回的值为偶数整数。

如果 *numeric-value* 并非空序列，并且 *precision* 并非为 0，那么返回的值是小数点右边有 *precision* 位并且接近 *numeric-value* 的数值。如果 *numeric-value* 与两个值的接近程度相等，那么返回的值为其最低有效数字为偶数的值。

返回值的数据类型取决于 *numeric-value* 的数据类型:

- 如果 *numeric-value* 的数据类型为 xs:float、xs:double、xs:decimal 或 xs:integer，那么返回的值的类型与 *numeric-value* 相同。
- 如果 *numeric-value* 的数据类型派生自 xs:float、xs:double、xs:decimal 或 xs:integer，那么返回值的数据类型为 *numeric-value* 的父代数据类型。

如果 *numeric-value* 是空序列，那么返回的值是空序列。

示例

没有精度自变量的示例: 以下函数返回 0.5 的舍入值:

```
fn:round-half-to-even(0.5)
```

返回的值为 0。

使用非零精度自变量的示例: 以下函数返回 1.5432，舍入为两位小数。

```
fn:round-half-to-even(1.5432,2)
```

返回的值为 1.54。

使用负精度的示例: 以下函数返回 35600。

```
fn:round-half-to-even(35612.25, -2)
```

seconds-from-dateTime 函数

fn:seconds-from-dateTime 函数返回 xs:dateTime 值的秒数部分。

语法

dateTime-value

要从中抽取秒数部分的 dateTime 值。

dateTime-value 的类型为 xs:dateTime，或者是空序列。

返回的值

如果 *dateTime-value* 的类型为 xs:dateTime，那么返回的值类型为 type xs:decimal，并且该值大于或等于 0 并小于 60。该值是 *dateTime-value* 的秒数部分和小数秒数部分。

如果 *dateTime-value* 是空序列，那么返回的值是空序列。

示例

以下函数返回 dateTime 值 2005 年 2 月 8 日下午 2:16:23（UTC-8 时区）的秒数部分。

```
fn:seconds-from-dateTime(xs:dateTime("2005-02-08T14:16:23-08:00"))
```

返回的值为 23。

以下函数返回 dateTime 值 2005 年 6 月 23 日上午 9:16:20.43（UTC 时区）的秒数部分。

```
fn:seconds-from-dateTime(xs:dateTime("2005-06-23T09:16:23.43Z"))
```

返回的值为 20.43。

seconds-from-duration 函数

fn:seconds-from-duration 函数返回持续时间的秒数部分。

语法

duration-value

要从中抽取秒数部分的持续时间值。

duration-value 是空序列，或者是下列其中一个类型的值：xdt:dayTimeDuration、xs:duration 或 xdt:yearMonthDuration。

返回的值

返回值取决于 *duration-value* 的类型：

- 如果 *duration-value* 的类型为 xdt:dayTimeDuration 或 xs:duration，那么返回的值类型为 xs:decimal，并且是大于 -60 并小于 60 的值。该值是转换为 xdt:dayTimeDuration 的 *duration-value* 的秒数部分和小数秒数部分。如果 *duration-value* 为负值，那么该值为负值。
- 如果 *duration-value* 的类型为 xdt:yearMonthDuration，那么返回的值类型为 xs:integer 并且值为 0。
- 如果 *duration-value* 是空序列，那么返回的值是空序列。

转换为 xdt:dayTimeDuration 的 *duration-value* 的秒数部分和小数秒数部分是按 (*S* mod 60) 计算的。值 *S* 是转换为 xdt:dayTimeDuration 的 *duration-value* 的秒和小数秒的总和，用于除去年数和月数部分。

示例

以下函数返回持续时间 150.5 秒的秒数部分。

```
fn:seconds-from-duration(xdt:dayTimeDuration("PT150.5S"))
```

返回的值为 30.5。计算持续时间的总秒数时，150.5 秒将转换为 2 分 30.5 秒。持续时间等于 PT2M30.5S，它的秒数部分为 30.5 秒。

seconds-from-time 函数

fn:seconds-from-time 函数返回 xs:time 值的秒数部分。

语法

time-value

要从中抽取秒数部分的时间值。

time-value 的类型为 xs:time，或者是空序列。

返回的值

如果 *time-value* 的类型为 xs:time，那么返回的值类型为 type xs:decimal，并且该值大于或等于 0 并小于 60。该值是 *time-value* 的秒数部分和小数秒数部分。

如果 *time-value* 是空序列，那么返回的值是空序列。

示例

以下函数返回时间值上午 08:59:59（UTC-8 时区）的秒数部分。

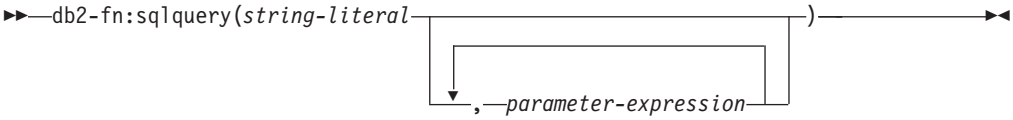
```
fn:seconds-from-time(xs:time("08:59:59-08:00"))
```

返回的值为 59。

sqlquery 函数

db2-fn:sqlquery 函数在当前相连 DB2 数据库中检索 SQL 全查询生成的序列。

语法



string-literal

包含全查询。全查询必须指定单列结果集，并且该列的数据类型必须为 XML。全查询的作用域是新的 SQL 查询作用域，而不是嵌套 SQL 查询。

全查询不能包含 isolation 子句或 lock-request 子句。

如果全查询包含单引号（例如，在字符串常量两边），那么应将函数自变量括在双引号中。例如:

```
"select c1 from t1 where c2 = 'Hello'"
```

如果全查询包含双引号（例如，在定界标识两边），那么应将函数自变量括在单引号中。例如:

```
'select c1 from "t1" where c2 = 47'
```

如果全查询同时包含单引号和双引号，那么应将函数自变量括在单引号中，并用两个相邻单引号字符来表示每个内部单引号。例如:

```
'select c1 from "t1" where c2 = ''Hello'''
```

全查询可包含对 PARAMETER 函数的调用，以引用 db2-fn:sqlquery 函数调用中指定的每个 *parameter-expression* 的结果值。在执行全查询时，PARAMETER 函数调用将替换为对应 *parameter-expression* 的结果值。

parameter-expression

会返回值的 XQuery 表达式。可通过指定的 SQL 函数 PARAMETER 及 SQL 全查询中的整数值自变量来引用每个 *parameter-expression* 的结果值。该整数值是 *parameter-expression* 的索引，指示它在 db2-fn:sqlquery 函数调用中的位置。有效整数值在 1 到函数调用中的 *parameter-expression* 总数之间。例如，如果 *string-literal* 自变量在 SQL 全查询中包含 `PARAMETER(1)` 和 `PARAMETER(2)`，那么函数调用必须指定两个 XQuery *parameter-expression* 自变量。`PARAMETER(1)` 引用第一个 *parameter-expression* 自变量的结果，而 `PARAMETER(2)` 引用第二个 *parameter-expression* 自变量的结果。

在处理 SQL 全查询期间，每个 PARAMETER 函数调用将替换为 db2-fn:sqlquery 函数调用中对应 *parameter-expression* 的结果值。每个 *parameter-expression* 只求值一次，不管在 SQL 全查询中引用多少次都是如此。

根据 XMLCAST 的规则，对应 *parameter-expression* 的结果数据类型必须可转换为 PARAMETER 函数的结果类型。否则会返回错误。

PARAMETER 函数的结果类型将按 SQL 全查询中的参数标记方式处理。例如，在其他上下文中，参数标记由问号（?）或者由一个冒号后跟名称（:*name*）来指示。如果不能确定 PARAMETER 函数的结果类型，那么会返回错误。

提示: 如果不允许在运算中使用隐式类型参数标记，那么可使用 CAST 规范或 XMLCAST 规范来指定类型。例如，要将 `PARAMETER(1)` 的强制类型转换为 DOUBLE，请使用以下 CAST 规范: `CAST(PARAMETER(1) as double)`。

返回的值

返回的值是 *string-literal* 中的全查询生成的序列。该全查询将作为 SQL 语句处理，遵循授权和名称解析的常规动态 SQL 规则。如果全查询包含对 PARAMETER 函数的任何调用，那么在对全查询求值时，这些调用会被替换为对应 *parameter-expression* 自变量的 XQuery 表达式的结果值。全查询返回的 XML 值将会并置以形成函数结果。包含空值的行不会影响结果序列。如果全查询未返回任何行或仅返回空值，那么函数的结果是空序列。

db2-fn:sqlquery 函数返回的序列中的项数可能与全查询返回的行数不同，原因是其中某些行可能包含空值或带有多项的序列。

示例

返回 XML 文档序列的全查询的示例: 以下示例显示了一些函数调用，它们从表 PRODUCT 中返回相同的文档序列。这些文档在列 DESCRIPTION。

下列所有函数将产生相同的结果:

```
db2-fn:sqlquery('select description from product')
db2-fn:sqlquery('SELECT DESCRIPTION FROM PRODUCT')
db2-fn:sqlquery('select "DESCRIPTION" from "PRODUCT"')
```

返回单个 XML 文档的全查询的示例: 以下示例返回包含表 PRODUCT 中的单个文档的序列。该文档在列 DESCRIPTION 中，并且由列 PID 的值“100-103-01”标识。

下列所有函数将产生相同的结果:

```
db2-fn:sqlquery('select Description from Product where pID=''100-103-01''')
db2-fn:sqlquery("select description from product where pid='100-103-01'")
db2-fn:sqlquery("select ""DESCRIPTION"" from product where pid='100-103-01'")
```

使用两个 PARAMETER 函数调用和一个表达式的全查询的示例: 以下示例返回推广日期内售出的所有部件的购买标识、部件标识和购买时间。

```
xquery
for $po in db2-fn:xmlcolumn('PURCHASEORDER.PORDER')/PurchaseOrder,
  $item in $po/item/partid
for $p in db2-fn:sqlquery(
  "select description
  from product
  where promostart < parameter(1)
  and  promoend  >  parameter(1)",
  $po/@OrderDate )
where $p//@pid = $item
return
<RESULT>
  <PoNum>{data($po/@PoNum)}</PoNum>
  <PartID>{data($item)} </PartID>
  <PoDate>{data($po/@OrderDate)}</PoDate>
</RESULT>
```

在处理 db2-fn:sqlquery 函数期间，对 parameter(1) 的两次引用会返回订单日期属性 $po/@OrderDate 的值。

使用两个 PARAMETER 函数调用和两个表达式的全查询的示例: 以下示例使用 DB2 SAMPLE 数据库中的 PURCHASEORDER 表。XQuery 表达式将检索订单日期在 2006 年 4 月 4 日之前的未交付订单，并列示每个订单的不同部件号:

```
  xquery
let $status := ( "Unshipped" ), $date := ( "2006-04-04" )
for $myorders in db2-fn:sqlquery(
   "select porder from purchaseorder
   where status = parameter(1)
   and orderdate < parameter(2)",
   $status, $date )
return
<LateOrder>
  <PoNum>
  {data($myorders/PurchaseOrder/@PoNum)}
  </PoNum>
  <PoDate>
  {data($myorders/PurchaseOrder/@OrderDate)}
  </PoDate>
  <Items>
     {for $itemID in distinct-values( $myorders/PurchaseOrder/item/partid )
     return
     <PartID>
     {$itemID}
     </PartID>}
   </Items>
</LateOrder>
```

在处理 db2-fn:sqlquery 函数期间，对 parameter(1) 的引用会返回表达式 $status 的结果集，而对 parameter(2) 的引用会返回表达式 $date 的结果值。

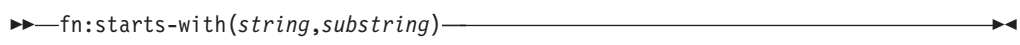
对 SAMPLE 数据库运行时，该表达式将返回以下结果:

```
<LateOrder>
  <PoNum>5000</PoNum>
  <PoDate>2006-02-18</PoDate>
  <Items>
    <PartID>100-100-01</PartID>
    <PartID>100-103-01</PartID>
  </Items>
</LateOrder>
```

starts-with 函数

fn:starts-with 函数确定字符串是否以特定子串开头。使用缺省整理来匹配搜索字符串。

语法



string 用于搜索 *substring* 的字符串。

string 的数据类型为 xs:string，或者是空序列。如果 *string* 是空序列，那么 *string* 设置为零长度字符串。

substring

用于在 *string* 开头进行搜索的子串。

substring 的数据类型为 xs:string，或者是空序列。

对于长度的限制

substring 的长度不能超过 32000 个字节。

返回的值

如果满足下列任一条件，那么返回的值为 xs:boolean 值 true:

- *substring*出现在 *string* 的开头。
- *substring* 是包含零长度字符串的空序列。

否则，返回的值为 false。

示例

以下函数确定字符串“Test literal”是否以字符串“lite”开头。

```
fn:starts-with('Test literal','lite')
```

返回的值为 false。

string 函数

fn:string 函数返回某个值的字符串表示。

语法

值　　要表示为字符串的值。

value 是节点或原子值，或者是空序列。

如果未指定 *value*，那么会为当前上下文项对 fn:string 求值。如果未定义当前上下文项，那么会返回错误。

返回的值

如果 *value* 并非空序列:

- 如果 *value* 是节点，那么返回的值是节点的字符串值。
- 如果 *value* 是原子值，那么返回的值是将 *value* 的强制类型转换为 xs:string 的结果。

如果 *value* 是空序列，那么结果为零长度字符串。

示例

以下函数返回 123 的字符串表示:

```
fn:string(xs:integer(123))
```

返回的值为“123”。

string-join 函数

fn:string-join 函数返回通过并置各项并用分隔符进行分隔而生成的字符串。

语法

sequence

要并置以形成字符串的项序列。

sequence 是任意 xs:string 值序列，或者是空序列。

separator

一个定界符，用于在 *sequence* 中的各项之间插入以形成字符串。

separator 的数据类型为 xs:string。

返回的值

返回的值是一个字符串，它是 *sequence* 中用 *separator* 隔开的各项的并置。如果 *separator* 是零长度字符串，那么 *sequence* 中的各项将在不使用分隔符的情况下并置。如果 *sequence* 是空序列，那么会返回零长度字符串。

示例

以下函数返回一个字符串，它是通过使用空格字符作为分隔符来并置序列 (″I″, ″made″, ″a″, ″sentence!″) 中的各项生成的。

```
fn:string-join(("I" , "made", "a", "sentence!"), " ")
```

返回的值是字符串“I made a sentence!”。

string-length 函数

fn:string-length 函数返回字符串的长度。

语法

```
►►—fn:string-length(source-string)———————————————————————————►◄
```

source-string
要对其返回长度的字符串。

source-string 的数据类型为 xs:string，或者是空序列。

返回的值

如果 *source-string* 并非空序列，那么返回的值为 *source-string* 的长度（以字符计）。xFFFF 以后的代码点使用两位 16 位值（又称为代理对）并且在计算字符串长度时被当成一个字符。*source-string* 是 xs:integer 值。

如果 *source-string* 是空序列，那么返回的值为 0。

示例

以下函数返回字符串“Test literal”的长度。

```
fn:string-length('Test literal')
```

返回的值为 12。

string-to-codepoints 函数

fn:string-to-codepoints 函数返回对应于字符串值的 Unicode 代码点序列。

语法

```
►►—fn:string-to-codepoints(source-string)———————————————————————►◄
```

source-string
要对其返回每个字符的 Unicode 代码点的字符串值，或者是空序列。

返回的值

如果 *source-string* 并非空序列，长度不为零，那么返回的值为 xs:integer 值序列，这些值表示 *source-string* 中的字符的代码点。

如果 *source-string* 是空序列并且长度为零，那么返回的值是空序列。

示例

以下函数返回代码点序列，这些代码点表示字符串“XQuery”中的字符。

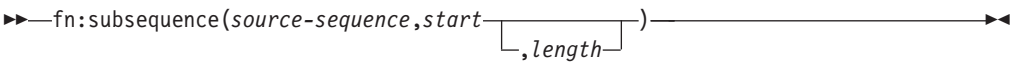
```
fn:string-to-codepoints("XQuery")
```

返回的值为 (88,81,117,101,114,121)。

subsequence 函数

fn:subsequence 函数返回序列的子序列。

语法



source-sequence
从中检索子序列的序列。

source-sequence 是任意序列，包括空序列。

start 子序列在 *source-sequence* 中的起始位置。*source-sequence* 的第一个位置为 1。如果 *start*<=0，那么 *start* 设置为 1。

start 的数据类型为 xs:double。

length 子序列中的项数。*length* 的缺省值是 *source-sequence* 中的项数。如果 *start*+*length*-1 大于 *source-sequence* 的长度，那么 *length* 设置为 (*source-sequence* 的长度)-*start*+1。

length 的数据类型为 xs:double。

返回的值

如果 *source-sequence* 并非空序列，那么返回的值是 *source-sequence* 的子序列，其起始位置为 *start* 并且包含 *length* 项。

如果 *source-sequence* 是空序列，那么会返回空序列。

示例

以下函数返回序列 ('T','e','s','t',' ','s','e','q','u','e','n','c','e') 中从第 6 项开始的 3 项。

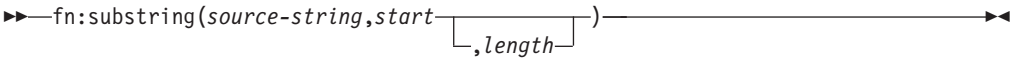
```
fn:subsequence(('T','e','s','t',' ','s','e','q','u','e','n','c','e'),6,3)
```

返回的值为 ('s','e','q')。

substring 函数

fn:substring 函数返回字符串的子串。

语法



source-string
要从中检索子串的字符串。

source-string 的数据类型为 xs:string，或者是空序列。

start 子串在 *source-string* 中的起始字符位置。*source-string* 的第一个位置为 1。如

果 *start*<= 0，那么 *start* 设置为 1。xFFFF 以后的代码点使用两位 16 位值（又称为代理对）并且在计数时被当成一个字符。

start 的数据类型为 xs:double。

length 子串的长度（以字符计）。*length* 的缺省值是 *source-string* 的长度。如果 *start*+*length*-1 大于 *source-string* 的长度，那么 *length* 设置为（*source-string* 的长度）-*start*+1。xFFFF 以后的代码点使用两位 16 位值（又称为代理对）并且在字符串长度方面计数为一个字符。

length 的数据类型为 xs:double。

返回的值

如果 *source-string* 并非空序列，那么返回的值是 *source-string* 的子串，其起始字符位置为 *start*，并且包含 *length* 个字符。如果 *source-string* 是空序列，那么会生成零长度字符串。

示例

以下函数返回起始于字符串“Test literal”的第 6 个字符的 7 个字符。

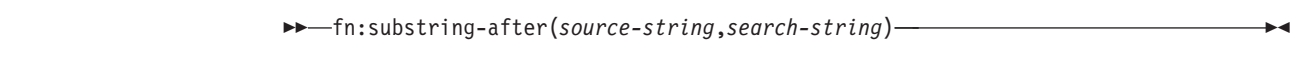
```
fn:substring('Test literal',6,7)
```

返回的值为“literal”。

substring-after 函数

fn:substring-after 函数返回字符串中位于第一次出现的特定搜索字符串后面的子串。使用缺省整理来匹配搜索字符串。

语法



source-string

要从中检索子串的字符串。

source-string 的数据类型为 xs:string，或者是空序列。如果 *source-string* 是空序列，那么 *source-string* 设置为零长度字符串。

search-string

一个字符串，将搜索它在 *source-string* 中第一次出现的位置。

search-string 的数据类型为 xs:string，或者是空序列。

对于长度的限制

search-string 的长度不能超过 32000 个字节。

返回的值

如果 *source-string* 并非空序列或零长度字符串:

- 假定 *source-string* 的长度为 *n*，并且 *m*<*n*。如果在 *source-string* 中找到 *search-string*，并且 *search-string* 在 *source-string* 中第一次出现位置的结尾位置为 *m*，那么返回的值是 *source-string* 中开始于位置 *m*+1 结束于位置 *n* 的子串。

- 假定 *source-string* 的长度为 *n*。如果在 *source-string* 中找到 *search-string*，并且 *search-string* 在 *source-string* 中第一次出现位置的结尾位置为 *n*，那么返回的值为零长度字符串。
- 如果 *search-string* 是空字符串或零长度字符串，那么返回的值为 *source-string*。
- 如果在 *source-string* 中找不到 *search-string*，那么返回的值为零长度字符串。

如果 *source-string* 为空序列或零长度字符串，那么返回的值为零长度字符串。

示例

以下函数使用缺省整理在字符串“DEFABCD”中找到“ABC”之后的字符。

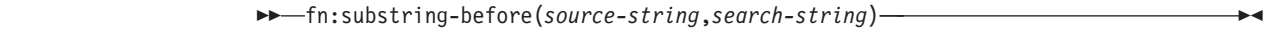
```
fn:substring-after('DEFABCD', 'ABC')
```

返回的值为“D”。

substring-before 函数

fn:substring-before 函数返回字符串中位于第一次出现的特定搜索字符串前面的子串。使用缺省整理来匹配搜索字符串。

语法



source-string
要从中检索子串的字符串。

source-string 的数据类型为 xs:string，或者是空序列。如果 *source-string* 是空序列，那么 *source-string* 设置为零长度字符串。

search-string
一个字符串，将搜索它在 *source-string* 中第一次出现的位置。

search-string 的数据类型为 xs:string，或者是空序列。

对于长度的限制

search-string 的长度不能超过 32000 个字节。

返回的值

如果 *source-string* 并非空序列或零长度字符串:

- 如果在 *source-string* 的位置 *m* 处找到 *search-string*，并且 *m*>1，那么返回的值是 *source-string* 中开始于位置 1 结束于位置 *m* 的子串。
- 如果在 *source-string* 中的位置 1 处找到 *search-string*，那么返回的值为零长度字符串。
- 如果 *search-string* 为空序列或零长度字符串，那么返回的值为零长度字符串。
- 如果在 *source-string* 中找不到 *search-string*，那么返回的值为零长度字符串。

如果 *source-string* 为空序列或零长度字符串，那么返回的值为零长度字符串。

示例

以下函数使用缺省整理在字符串“DEFABCD”中找到“ABC”之前的字符。

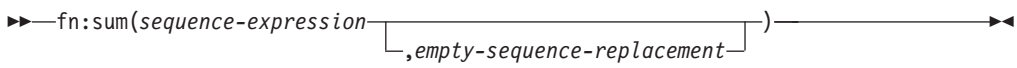
```
fn:substring-before('DEFABCD', 'ABC')
```

返回的值为“DEF”。

sum 函数

fn:sum 函数返回序列中的值的总和。

语法



sequence-expression

包含下列任一原子类型的项序列或空序列:

- xs:float
- xs:double
- xs:decimal
- xs:integer
- xdt:untypedAtomic
- xdt:dayTimeDuration
- xdt:yearMonthDuration
- 从上面列示的任一类型派生的类型

类型为 xdt:untypedAtomic 的输入项将转换为 xs:double。此次强制类型转换后，输入序列中的所有项必须可通过提升或子类型替换从而转换为公共类型。总和是使用此公共类型计算的。例如，如果输入序列包含类型为 money（派生自 xs:decimal）和 stockprice（派生自 xs:float）的项，那么总和将使用类型 xs:float 进行计算。

empty-sequence-replacement

sequence-expression 为空序列时返回的值。*empty-sequence-replacement* 可以具有对 *sequence-expression* 列示的一种数据类型。

返回的值

如果 *sequence-expression* 并非空序列，那么返回的值是 *sequence-expression* 中的值的总和。返回值的数据类型与 *sequence-expression* 中各项的数据类型相同，或者是 *sequence-expression* 中各项提升至的数据类型。

如果 *sequence-expression* 是空序列，并且未指定 *empty-sequence-replacement*，那么 fn:sum 会返回 0.0E0。如果 *sequence-expression* 是空序列，并且指定了 *empty-sequence-replacement*，那么 fn:sum 会返回 *empty-sequence-replacement*。

示例

以下函数返回序列 (500, 1.0E2, 40.5) 之和:

```
fn:sum((500, 1.0E2, 40.5))
```

这些值将提升至 xs:double 数据类型。该函数返回 xs:double 值 6.405E2，它将以序列化方式表示为“640.5”。

timezone-from-date 函数

fn:timezone-from-date 函数返回 xs:date 值的时区部分。

语法

date-value

要从中抽取时区部分的日期值。

date-value 的类型为 xs:date，或者为空序列。

返回的值

如果 *date-value* 的类型为 xs:date 并且具有显式时区部分，那么返回的值类型为 xdt:dayTimeDuration，并且值的范围在 -PT14H 到 PT14H 之间（包含两者）。该值是 UTC 时区与 *date-value* 时区部分的偏差。

如果 *date-value* 没有显式时区部分，或者为空序列，那么返回的值是空序列。

示例

以下函数返回日期值 2007 年 3 月 13 日（UTC-8 时区）的时区部分。

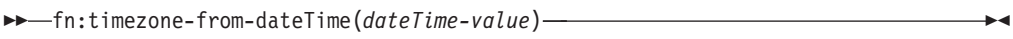
```
fn:timezone-from-date(xs:date("2007-03-13-08:00"))
```

返回的值为 -PT8H。

timezone-from-dateTime 函数

fn:timezone-from-dateTime 函数返回 xs:dateTime 值的时区部分。

语法



dateTime-value

要从中抽取时区部分的 dateTime 值。

dateTime-value 的类型为 xs:dateTime，或者是空序列。

返回的值

如果 *dateTime-value* 的类型为 xs:dateTime 并且具有显式时区部分，那么返回的值类型为 xdt:dayTimeDuration，并且值的范围在 -PT14H 到 PT14H 之间（包含两者）。该值是 UTC 时区与 *dateTime-value* 时区部分的偏差。

如果 *dateTime-value* 没有显式时区部分，或者为空序列，那么返回的值是空序列。

示例

以下函数返回 dateTime 值 2005 年 10 月 30 日上午 7:30（UTC-8 时区）的时区部分。

```
fn:timezone-from-dateTime(xs:dateTime("2005-10-30T07:30:00-08:00"))
```

返回的值为 -PT8H。

以下函数返回 dateTime 值 2005 年 1 月 1 日下午 2:30（UTC+10:30 时区）的时区部分。

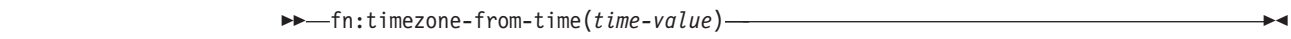
```
fn:timezone-from-dateTime(xs:dateTime("2005-01-01T14:30:00+10:30"))
```

返回的值为 PT10H30M。

timezone-from-time 函数

fn:timezone-from-time 函数返回 xs:time 值的时区部分。

语法



time-value

要从中抽取时区部分的时间值。

time-value 的类型为 xs:time，或者是空序列。

返回的值

如果 *time-value* 的类型为 xs:time 并且具有显式时区部分，那么返回的值类型为 xdt:dayTimeDuration，并且值的范围在 -PT14H 到 PT14H 之间（包含两者）。该值是 UTC 时区与 *time-value* 时区部分的偏差。

如果 *time-value* 没有显式时区部分，或者为空序列，那么返回的值是空序列。

示例

以下函数返回时间值中午 12 点（UTC-5 时区）的时区部分。

```
fn:timezone-from-time(xs:time("12:00:00-05:00"))
```

返回的值为 -PT5H。

在以下函数中，时间值下午 1:00 没有时区部分。

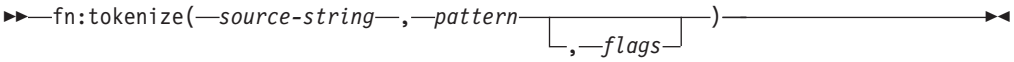
```
fn:timezone-from-time(xs:time("13:00:00"))
```

返回的值是空序列。

tokenize 函数

fn:tokenize 函数将字符串分为一系列子串。

语法



source-string

要分割成子串序列的字符串。

source-string 是 xs:string 值或空序列。

pattern *source-string* 中的子串之间的定界符。

pattern 是包含正则表达式的 xs:string 值。正则表达式是用于在搜索模式中定义一个或一组字符串的字符、通配符和运算符集合。

flags 一个 xs:string 值，可包含下列用于控制 *pattern* 与 *source-string* 中字符的匹配情况的任何值:

s 指示正则表达式中的点（.）与包括换行符 X'0A' 在内的任意字符相匹配

如果未指定标志，那么点（.）与换行符（X'0A'）以外的任意字符相匹配。

m 指示插入标记（^）与行的开头（换行符之后的位置)相匹配，而美元符号（$）与行的结尾（换行符之前的位置）相匹配。

如果未指定 m 标志，那么插入标记（^）与字符串的开头相匹配，并且美元符号（$）与字符串的结尾相匹配。

i 标识匹配是不区分大小写的。

如果未指定 i 标志，那么会执行区分大小写匹配。

x 指示 *pattern* 中的空格字符会被忽略。

如果未指定 x 标志，那么空格字符会用于匹配。

对于长度的限制

source-string 和 *pattern* 的长度不能超过 32000 个字节。

返回的值

如果 *source-string* 并非空序列或零长度字符串，那么返回的值是对 *source-string* 执行下列操作时生成的序列:

- 将搜索 *source-string* 以查找与 *pattern* 相匹配的字符。
- 如果 *pattern* 包含两组或更多备用字符集，那么与 *source-string* 中字符相匹配的 *pattern* 中的第一组字符被视为匹配模式。
- 与 *pattern* 不匹配的每组字符将成为结果序列中的一项。
- 如果 *pattern* 与 *source-string* 开头的字符相匹配，那么返回序列中的第一项是零长度字符串。
- 如果 *source-string* 中发现对应 *pattern* 的两个连续匹配项，那么会向序列添加零长度字符串。
- 如果 *pattern* 与 *source-string* 结尾的字符相匹配，那么返回序列中的最后一项是零长度字符串。

如果在 *source-string* 中找不到 *pattern*，那么会返回错误。

如果 *source-string* 是空序列或零长度字符串，那么结果为空序列。

示例

以下函数通过字符串“Tokenize this sentence, please.”创建序列。“\s+”是一个正则表达式，用于指示一个或多个字符。

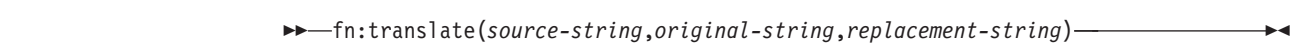
```
fn:tokenize("Tokenize this sentence, please.", "\s+")
```

返回的值是序列 (″Tokenize″, ″this″, ″sentence,″, ″please.″)。

translate 函数

fn:translate 函数将字符串中的所选字符替换为替换字符。

语法



source-string

要转换其中的字符的字符串。

source-string 的数据类型为 xs:string，或者是空序列。

original-string

包含可转换的字符的字符串。

original-string 的数据类型为 xs:string。

replacement-string

一个字符串，它包含的字符将替换 *original-string* 中的字符。

replacement-string 的数据类型为 xs:string。

如果 *replacement-string* 的长度大于 *original-string* 的长度，那么 *replacement-string* 中的多余字符会被忽略。

对于长度的限制

original-string 和 *replacement-string* 的长度不能超过 32000 个字节。

返回的值

如果 *source-string* 并非空序列，那么返回的值是执行下列操作时生成的 xs:string 值:

- 对于 *source-string* 中出现在 *original-string* 中的每个字符，应将 *source-string* 中的字符替换为 *replacement-string* 中的对应字符，该字符的出现位置与 *original-string* 中的字符出现位置相同。使用二进制比较来匹配字符。

  如果 *original-string* 的长度大于 *replacement-string* 的长度，那么应在 *source-string* 中删除出现在 *original-string* 中的每个字符，但 *original-string* 中的字符位置不必与 *replacement-string* 中的字符位置相对应。

  如果某个字符在 *original-string* 中出现多次，那么该字符在 *original-string* 中第一次出现的位置将确定 *replacement-string* 中使用的字符。

- 对于 *source-string* 中未出现在 *original-string* 中的每个字符，应将该字符保留原状。

如果 *source-string* 是空序列，那么会返回零长度字符串。

示例

以下函数返回在字符串“Test literal”中将 e 替换为 o 并将 l 替换为 m 而生成的字符串。

```
fn:translate('Test literal','el','om')
```

返回的值为“Tost mitoram”。

以下函数返回在字符串文字“Another test literal”中进行以下替换后生成的字符串：A 替换为 B，t 替换为 f，e 替换为 i 并且 r 替换为 m。

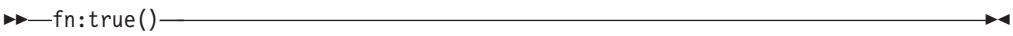
```
fn:translate('Another test literal', 'Ater', 'Bfim')
```

返回的值为“Bnofhim fisf lifimal”。

true 函数

fn:true 函数返回 xs:boolean 值 true。

语法



返回的值

返回的值为 xs:boolean 值 true。

示例

使用 true 函数以返回值 true。

```
fn:true()
```

返回的值为 true。

unordered 函数

fn:unordered 函数以不确定的顺序返回序列中的项。

语法

sequence-expression
任何序列，包括空序列。

返回的值

返回的值是 *sequence-expression* 中按不确定顺序排列的项。这会帮助查询优化器选择独立于序列中各项排列顺序的存取路径。

示例

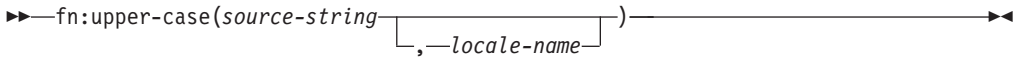
以下函数以不确定的顺序返回序列 (1,2,3) 中的各项。

```
fn:unordered((1,2,3))
```

upper-case 函数

fn:upper-case 函数将字符串转换为大写。

语法



source-string

要转换为大写的字符串。

source-string 的数据类型为 xs:string，或者是空序列。

locale-name

包含要用于大写操作的语言环境的字符串。

locale-name 的类型为 xs:string，或者是空序列。如果 *locale-name* 并非空序列，那么 *locale-name* 的值是不区分大小写的，并且必须是有效语言环境或零长度字符串。

返回的值

如果 *source-string* 并非空序列，那么返回的值为 *source-string*，并且每个字符会转换为对应大写。如果 *locale-name* 未指定、为空序列或者是零长度字符串，那么会使用 Unicode 标准中定义的大写规则。否则会使用指定语言环境的大写规则。没有对应大写形式的每个字符将以其原始形式包含在返回的值中。

如果 *source-string* 是空序列，那么返回的值是零长度字符串。

示例

以下函数会将字符串“Test literal 1”转换为大写。

```
fn:upper-case('Test literal 1')
```

返回的值为“TEST LITERAL 1”。

以下函数指定土耳其语言环境 tr_TR 并转换字母“i”及数字字符引用 ı（拉丁小写字母无点 i 的字符引用）。

```
fn:upper-case("i&#x131;", "tr_TR")
```

返回的值由两个字符组成: İ 表示的字符（拉丁大写 I，上面有点）和字母“I”。对于土耳其语言环境，字母“i”将转换为 İ 表示的字符（拉丁大写 I，上面有点），而 ı 表示的字符（拉丁小写字母无点 i）将转换为字母“I”。

以下函数未指定语言环境，并使用 Unicode 标准中定义的规则将两个字符转换为大写。

```
fn:upper-case("&#x131;i")
```

该函数返回字符“II”。fn:upper-case 将小写字符 ı 和字母“i”转换为大写字母“I”。

xmlcolumn 函数

db2-fn:xmlcolumn 函数在当前相连 DB2 数据库中检索某列中的序列。

语法

```
►►—db2-fn:xmlcolumn(string-literal)————————————————————►◄
```

string-literal

指定要从中检索序列的列的名称。列名必须用表名、视图名或别名限定，并且必须引用使用 XML 数据类型的列。SQL 模式名称是可选的。如果未指定 SQL 模式名，那么 CURRENT SCHEMA 专用寄存器会用作表或视图的隐式限定符。*string-literal* 是区分大小写的。*string-literal* 必须使用用于在数据库中标识该列名的准确字符。

返回的值

返回的值是一个序列，它是 *string-literal* 指定的列中非空 XML 值的并置。如果表或视图中没有任何行，那么 db2-fn:xmlcolumn 会返回空序列。

db2-fn:xmlcolumn 函数返回的序列中的项数可能与指定表或视图中的行数不同，原因是其中某些行可能包含空值或带有多项的序列。

db2-fn:xmlcolumn 函数与 db2-fn:sqlquery 函数相关，并且两个函数可生成同一结果。但是，两个函数的自变量在区分大小写方面有所不同。db2-fn:xmlcolumn 函数中的自变量由 XQuery 处理，所以它是区分大小写的。因为 DB2 数据库中的表名和列名在缺省情况下是大写的，所以 db2-fn:xmlcolumn 的自变量通常使用大写。db2-fn:sqlquery 函数的自变量是由 SQL 处理的，SQL 会自动将标识转换为大写。

下列函数调用是等价的，并且会返回相同结果:

```
db2-fn:xmlcolumn('SQLSCHEMA.TABLENAME.COLNAME')
db2-fn:sqlquery('select colname from sqlschema.tablename')
```

示例

返回文档序列的示例: 以下函数返回 XML 文档序列，这些文档存储在表 PRODUCT 的 XML 列 DESCRIPTIONT 中，对于此示例，该表位于 SQL 模式 SAMPLE 中。

```
db2-fn:xmlcolumn('SAMPLE.PRODUCT.DESCRIPTION')
```

使用隐式 SQL 模式的示例: 在以下示例中，DB2 数据库中的 CURRENT SCHEMA 专用寄存器设置为 SAMPLE，所以该函数返回的结果与上一示例相同:

```
db2-fn:xmlcolumn('PRODUCT.DESCRIPTION')
```

使用 SQL 定界标识的示例: 以下函数返回存储在表“Student”的列“Thesis”中的文档序列，假定该表位于当前指定给 CURRENT SCHEMA 的模式中。因为表名和列名包含小写字符，所以可使用两种方式在 db2-fn:xmlcolumn 函数的字符串文字自变量中对其进行指定:

- 指定为 SQL 定界标识（括在双引号中）:

```
db2-fn:xmlcolumn('"Student"."Thesis"')
```

- 指定为字符串而不指示它们是 SQL 定界标识:

```
db2-fn:xmlcolumn('Student.Thesis')
```

相反，我们需要 db2-fn:sqlquery 函数中使用的相同表和列信息才能按如下方式使用 SQL 定界标识:

```
db2-fn:sqlquery('select "Thesis" from "Student"')
```

year-from-date 函数

fn:year-from-date 函数返回 xs:date 值的年份部分。

语法

```
►►—fn:year-from-date(date-value)———————————————————————————►◄
```

date-value

要从中抽取年份部分的日期值。

date-value 的类型为 xs:date，或者为空序列。

返回的值

如果 *date-value* 的类型为 xs:date，那么返回的值类型为 xs:integer，该值是 *date-value* 的年份部分，并且是非负值。

如果 *date-value* 是空序列，那么返回的值是空序列。

示例

以下函数返回日期值 2005 年 10 月 29 日的年份部分。

```
fn:year-from-date(xs:date("2005-10-29"))
```

返回的值为 2005。

year-from-dateTime 函数

fn:year-from-dateTime 函数返回 xs:dateTime 值的年份部分。

语法

```
►►—fn:year-from-dateTime(dateTime-value)———————————————————————►◄
```

dateTime-value

要从中抽取年份部分的 dateTime 值。

dateTime-value 的类型为 xs:dateTime，或者是空序列。

返回的值

如果 *dateTime-value* 的类型为 xs:dateTime，那么返回的值类型为 xs:integer。该值是 *dateTime-value* 的年份部分，并且是非负值。

如果 *dateTime-value* 是空序列，那么返回的值是空序列。

示例

以下函数返回 dateTime 值 2005 年 10 月 29 日上午 8:00（UTC-8 时区）的年份部分。

```
fn:year-from-dateTime(xs:dateTime("2005-10-29T08:00:00-08:00"))
```

返回的值为 2005。

years-from-duration 函数

fn:years-from-duration 函数返回持续时间的年数部分。

语法

```
►►—fn:years-from-duration(duration-value)———————————————————————►◄
```

duration-value

要从中抽取年数部分的持续时间值。

duration-value 是空序列，或者是下列其中一个类型的值：xdt:dayTimeDuration、xs:duration 或 xdt:yearMonthDuration。

返回的值

返回值取决于 *duration-value* 的类型：

- 如果 *duration-value* 的类型为 xdt:yearMonthDuration 或者为 xs:duration，那么返回的值类型为 xs:integer。该值是转换为 xdt:yearMonthDuration 的 *duration-value* 的年数部分。如果 *duration-value* 为负值，那么该值为负值。
- 如果 *duration-value* 的类型为 xs:dayTimeDuration，那么返回的值类型为 xs:integer 并且值为 0。
- 如果 *duration-value* 是空序列，那么返回的值是空序列。

转换为 xdt:yearMonthDuration 的 *duration-value* 年数部分是由转换为 xdt:yearMonthDuration 的 *duration-value* 总月数除以 12 确定的。

示例

以下函数返回持续时间 -4 年 -11 个月 -320 天的年数。

```
fn:years-from-duration(xs:duration("-P4Y11M320D"))
```

返回的值为 -4。

以下函数返回持续时间 9 年零 13 个月的年数部分。

```
fn:years-from-duration(xdt:yearMonthDuration("P9Y13M"))
```

返回的值为 10。计算持续时间的总年数时，13 个月将转换为 1 年零 1 个月。持续时间等于 P10Y1M，其年数部分为 10 年。

zero-or-one 函数

如果自变量包含一项或为空序列，那么 fn:zero-or-one 函数返回自变量。

语法

sequence-expression
任何序列，包括空序列。

返回的值

如果 *sequence-expression* 包含一项或者是空序列，那么会返回 *sequence-expression*。否则会返回错误。

示例

以下示例使用 fn:zero-or-one 函数来确定变量 $seq 中的序列是否包含一项或零项。

```
let $seq := (5,10)
return fn:zero-or-one($seq)
```

返回错误，原因是序列包含两项。

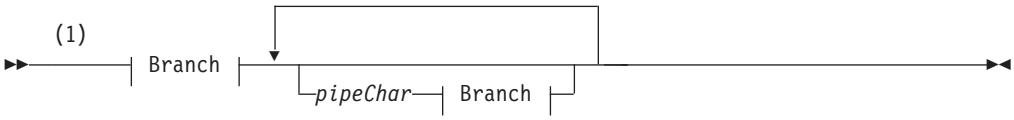
第 6 章 正则表达式

正则表达式是一个字符序列，这些字符充当匹配和操作字符串的模式。

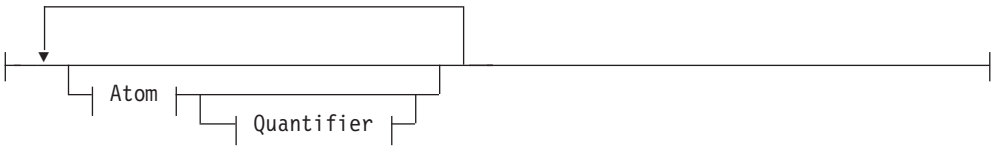
正则表达式用于下列 XQuery 函数: fn:matches、fn:replace 和 fn:tokenize。DB2 XQuery 正则表达式支持基于 W3C Recommendation *XML Schema Part 2: Datatypes Second Edition* 中定义的 XML 模式正则表达式支持以及 W3C Recommendation *XQuery 1.0 and XPath 2.0 Functions and Operators* 定义的扩展。

语法

RegularExpression

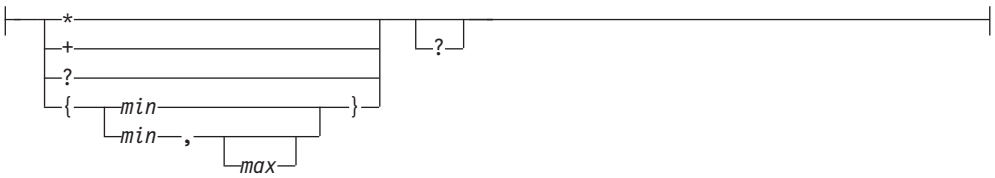


Branch:



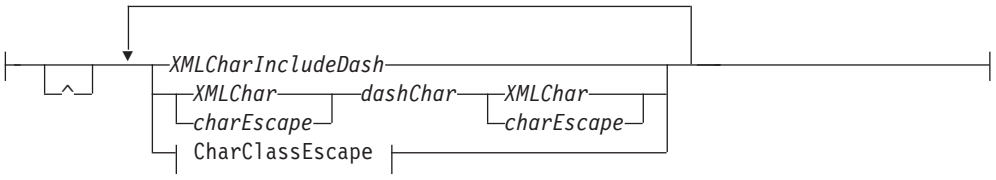
Atom:

Quantifier:



CharClassExpression:

CharGroup:



CharClassEscape:

注:

1 正则表达式的语法表示字符串文字内容，它不能包含具有作为模式字符的空格字符的特殊含义以外的空格字符。不要将语法元素之间的空格或部分视为允许使用任何形式的空格。

RegularExpression

正则表达式包含一个或多个分支。分支用管道（|）隔开，指示每个分支都是备用模式。

pipeChar
管道字符（|）会隔开正则表达式中的备用分支。

Branch
分支由零个或零个以上原子组成，每个原子允许一个可选量词。

Atom

原子是普通字符、字符类表达式、字符类转义或带括号正则表达式。

normalCharacter
并非第 198 页的表 38 中的任何匹配字符的任何有效 XML 字符。

^ 在分支开头使用时，插入标记（^）指示该模式必须从字符串开头匹配。

**$** 在分支结尾使用时，美元符号（$）指示该模式必须从字符串的结尾匹配。

Quantifier

量词指定原子在正则表达式的重复词数。缺省情况下，量词会使用称为*贪婪算法*的方式尽量与目标字符串进行匹配。例如，正则表达式 `'A.*A'` 与整个字符串 'ABACADA' 匹配，原因是必需的外部 'A' 字符之间的子串符合对任意字符的要求任意次数。可通过在量词后指定问号（?）字符来更改缺省贪婪算法。问号指定该模式匹配使用*勉强算法*，这会与目标字符串中满足正则表达式的下一个最短子串进行匹配（按从左至右的顺序）。例如，正则表达式 `'A.*?A'` 会与子串 'ABA' 和 'ADA' 匹配，而不是与整个

字符串 'ABACADA' 匹配。通过使用勉强算法与正则表达式匹配的子串字符不会进行更进一步的匹配。这就是 'ACA' 在上一示例中未被视为匹配项的原因。勉强算法与 fn:replace 函数一起使用时很有帮助，原因是它会按从左至右的顺序处理匹配和进行替换。

例如，如果在函数 `fn:replace("nonsensical","n(.*)s","mus")` 中使用贪婪算法将以“n”开头“s”结尾的字符串替换为字符串“mus”，那么返回的值为“musical”。原始字符串包括子串“nons”和“ns”，并且与按从左至右进行的模式扫描（以查找下一匹配项）相匹配，但贪婪算法不会处理这些匹配，原因是它发现了更长的封闭匹配。

如果对函数 `fn:replace("nonsensical","n(.*?)s","mus")` 中的同一字符串使用勉强算法，那么结果会有所不同。返回的值为“musemusical”。在此情况下字符串中会进行两次替换。第一个匹配将“nons”替换为“mus”，第二个匹配将“ns”替换为“mus”。

又例如，如果在函数 `fn:replace("AbrAcAdAbrA","A(.*)A","X$1X")` 中使用贪婪算法将任何数目的字符两旁的字符 A 替换为相同字符两旁的字符 X，那么返回的值为“XbrAcAdAbrX”。原始字符串包括子串“AbrA”和“AdA”，并且与从左至右扫描以查找下一匹配时的模式相匹配，但贪婪算法不会处理这些匹配，原因是它发现了更长的封闭匹配。

如果对函数 `fn:replace("AbrAcAdAbrA","A(.*?)A","X$1X")` 中的同一字符串使用勉强算法，那么结果会有所不同。返回的值为“XbrXcXdXbrA”。在此情况下，字符串会进行两次替换，第一次替换“AbrA”，第二次替换“AdA”。字符串最后的“A”不会被替换，原因是勉强算法将前面所有的“A”字符用于字符串中的其他匹配。不会考虑原始字符串中以字符“A”开头和结尾的其他子串（如“AcA”、“AcAdA”、“AdAbrA”和“AbrA”），原因是它们在参与模式匹配后勉强算法会将这些字符视为已使用。

`*` 与原子匹配零次或多次。相当于量词 `{0,}`。

`+` 与原子匹配一次或多次。相当于量词 `{1,}`。

`?` 与原子匹配零次或一次。相当于量词 `{0,1}`。如果跟在另一量词后，那么指示使用勉强算法而不是贪婪算法。

min

与原子匹配至少 *min* 次。*min* 必须为正整数。

- {*min*} 与原子正好匹配 *min* 次。
- {*min*,} 与原子匹配至少 *min* 次。

max

与原子匹配不超过 *max* 次。*max* 必须为大于或等于 *min* 的正整数。

- {0,*max*} 与原子匹配不超过 *min* 次。
- {0,0} 仅与空字符串匹配。

CharGroup

`^` 指示由 CharGroup 的余下部分定义的字符组的补充部分。

dashChar

虚线字符（-）会隔开用于定义一定字符范围中的外部字符的两个字符。格式为 `s-e` 的字符范围是一组 UCS2 代码点，这些代码点大于或等于 `s` 并且小于或等于 `e`:

- `s` 不是反斜杠（\）
- 如果 `s` 是 CharGroup 中的第一个字符，那么它不是插入标记（^）

- e 不是反斜杠（\）或左方括号（[）
- e 的代码点大于 s 的代码点

XMLCharIncludeDash

有效 XML 字符的单字符形式，排除反斜杠（\）和方括号（[]），但包括虚线字符（-）。虚线字符只有出现在 CharGroup 的开头或结尾才是有效字符。CharGroup 开头的插入标记（^）指示组的补充部分。插入标记出现在组中的任何其他位置时仅与插入标记相匹配。*XMLCharIncludeDash* 可包括根据正则表达式 [^\#5B#5D] 匹配的任何字符。

XMLChar

有效 XML 字符的单字符形式，排除反斜杠（\）、方括号（[]）和虚线字符（-）。虚线字符只有出现在 CharGroup 的开头或结尾才是有效字符。CharGroup 开头的插入标记（^）指示组的补充部分。插入标记出现在组中的任何其他位置时仅与插入标记相匹配。*XMLChar* 可包括根据正则表达式 [^\#2D#5B#5D] 匹配的任何字符。

charEscape

反斜杠，后跟单个元字符、换行符、回车符或跳进字符。必须在正则表达式中对表 38 中的字符转义以进行匹配。

表 38. 有效元字符转义

字符转义	表示的字符	描述
\n	#x0A	换行
\r	#x0D	回车
\t	#x09	制表符
\\	\	反斜杠
\\|	|	管道
\.	.	句点
\-	-	连字符
\^	^	插入标记
\?	?	问号
\$	$	美元符号
*	*	星号
\+	+	加号
\{	{	左花括号
\}	}	右花括号
\(	(	左圆括号
\)	)	右圆括号
\[	[	左方括号
\]	]	右方括号

CharClassEscape

. 句点字符（.）与换行符和回车符以外的所有字符相匹配。句点字符等价于表达式 [^\n\r]/

nonZeroDigit

指定与根据子表达式匹配的字符串相匹配的向后引用，该子表达式在正则表达式中的 *nonZeroDigit* 位置是用圆括号括起来的。*nonZeroDigit* 必须在 1 到 9 之间。可引用前 9 个子表达式。

注: 为了将来实现向上兼容性，如果向后引用后跟数字字符，那么应将向后引用括在圆括号中。例如，对后跟数字 3 的第一个子表达式的向后引用应表示为 (/1)3 而不是 /13，即使两者目前会生成同一结果也是如此。

P{**Is***blockName*}

指定 Unicode 代码点范围的补充部分。范围由 *blockName* 标识，如 *XML Schema Part 2: Datatypes Second Edition* 中列示的那样。

p{**Is***blockName*}

在特定范围的 Unicode 代码点中指定字符。范围由 *blockName* 标识，如 *XML Schema Part 2: Datatypes Second Edition* 中列示的那样。

charEscape

反斜杠，后跟单个元字符、换行符、回车符或跳进字符。必须在正则表达式中对第 198 页的表 38 中的字符转义以进行匹配。

multiCharEscape

一个反斜杠，后跟一个字符，用于在正则表达式中标识表 39 中的常用字符集以进行匹配。

表 39. 多字符转义

多字符转义	等价正则表达式	描述
\s	[#x20\t\n\r]	空格、制表符、换行或回车符。
\S	[^\s]	空格、制表符、换行或回车符以外的任何字符。
\i	无	允许作为 XML 名称的第一个字符的字符集。
\I	[^\i]	不在允许作为 XML 名称的第一个字符的字符集中。
\c	无	XML 名称中允许使用的字符集。
\C	[^\c]	在 XML 名称中不允许使用的字符集中。
\d	\p{Nd}	十进制数。
\D	[^\d]	非十进制数。
\w	[#x0000-#x10FFFF]-[\p{P}\p{Z}\p{C}]	单词字符，包括下列 *charProperty* 类别: 字母、标记、符号和数字。
\W	[^\w]	非单词字符，包括下列 *charProperty* 类别: 标点、分隔符及其他。

p{*charProperty*}

指定某个类别中的某个字符。类别列示在表 40 中。

P{*charProperty*}

指定字符类别的补充部分。类别列示在表 40 中。

表 40. charProperty 的受支持值

charProperty	类别	描述
L	字母	所有字母
Lu	字母	大写

表 40. charProperty 的受支持值（续）

charProperty	**类别**	**描述**
Ll	字母	小写
Lt	字母	标题
Lm	字母	修饰符
Lo	字母	其他
M	标记	所有标记
Mn	标记	非间隔
Mc	标记	间隔组合
Me	标记	封闭
N	数字	所有数字
Nd	数字	十进制数
Nl	数字	字母
否	数字	其他
P	标点	所有标点
Pc	标点	连接符
Pd	标点	连字符
Ps	标点	左
Pe	标点	右
Pi	标点	初始引号（根据用法，行为方式类似 Ps 或 Pe）
Pf	标点	最终引号（根据用法，行为方式类似 Ps 或 Pe）
Po	标点	其他
Z	分隔符	所有分隔符
Zs	分隔符	空格
Zl	分隔符	行
Zp	分隔符	段
S	符号	所有符号
Sm	符号	数学
Sc	符号	货币
Sk	符号	修饰符
So	符号	其他
C	其他	所有其他
Cc	其他	控制
Cf	其他	格式
Co	其他	专用
Cn	其他	未指定

注: 使用二进制比较来匹配正则表达式。不使用缺省整理。

第 7 章 限制

DB2 XQuery 存在大小和数据类型方面的限制。

XQuery 数据类型的限制

本主题标识特定 DB2 XQuery 数据类型允许的值的范围。

表 41. XQuery 数字数据类型的限制

数据类型	最小值	最大值	其他限制
xs:float	-3.4028234663852886e+38	+3.4028234663852886e+38	最小正值: +1.1754943508222875e-38 最大负值: -1.1754943508222875e-38
xs:double	-1.7976931348623158e+308	+1.7976931348623158e+308	最小正值: +2.2250738585072014e-308 最大负值: +2.2250738585072014e-308
xs:decimal	不可用	不可用	最大十进制精度: 31 位
xs:integer	-9 223 372 036 854 775 808	+9 223 372 036 854 775 807	
xs:nonPositiveInteger	-9 223 372 036 854 775 808	0	
xs:negativeInteger	-9 223 372 036 854 775 808	-1	
xs:long	-9 223 372 036 854 775 808	9 223 372 036 854 775 807	
xs:int	-2 147 483 648	+2 147 483 647	
xs:short	-32 768	+32767	
xs:byte	-128	+127	
xs:nonNegativeInteger	0	+9 223 372 036 854 775 807	
xs:unsignedLong	0	+9 223 372 036 854 775 807	
xs:unsignedInt	0	4 294 967 295	
xs:unsignedShort	0	+65 535	
xs:unsignedByte	0	+255	
xs:positiveInteger	+1	+9 223 372 036 854 775 807	

表 42. XQuery 日期、时间和持续时间数据类型的限制

数据类型	最小值	最大值
xs:duration	-P83333333333333Y3M11574074074DT1H46M39.999999S	P83333333333333Y3M11574074074DT1H46M39.999999S
xdt:yearMonthDuration	-P83333333333333Y3M	P83333333333333Y3M
xdt:dayTimeDuration	-P11574074074DT1H46M39.999999S	P11574074074DT1H46M39.999999S
xs:dateTime	0001-01-01T00:00:00.000000Z	9999-12-31T23:59:59.999999Z
xs:date	0001-01-01Z	9999-12-31Z
xs:time	00:00:00Z	23:59:59Z
xs:gDay	01Z	31Z

表 42. XQuery 日期、时间和持续时间数据类型的限制 (续)

数据类型	最小值	最大值
xs:gMonth	01Z	12Z
xs:gYear	0001Z	9999Z
xs:gYearMonth	0001-01Z	9999-12Z
xs:gMonthDay	01-01Z	12-31Z

注: DB2 XQuery 不支持负日期或负时间。

大小限制

DB2 XQuery 的字符串文字和查询存在大小限制。

字符串文字的大小限制为 32672 个字节。

查询长度的大小限制为 2 097 152 字节。

附录 A. DB2 技术信息概述

DB2 技术信息以多种可以通过多种方法访问的格式提供。

您可以通过下列工具和方法获得 DB2 技术信息:

- DB2 信息中心
  - 主题（任务、概念和参考主题）
  - 样本程序
  - 教程
- DB2 书籍
  - PDF 文件（可下载）
  - PDF 文件（在 DB2 PDF DVD 中）
  - 印刷版书籍
- 命令行帮助
  - 命令帮助
  - 消息帮助

注: DB2 信息中心主题的更新频率比 PDF 书籍或硬拷贝书籍的更新频率高。要获取最新信息，请安装可用的文档更新或者参阅 ibm.com 上的 DB2 信息中心。

您可以在线访问 ibm.com 上的其他 DB2 技术信息，例如技术说明、白皮书和 IBM Redbooks® 出版物。请访问以下网址处的 DB2 信息管理软件资料库站点: http://www.ibm.com/software/data/sw-library/。

文档反馈

我们非常重视您对 DB2 文档的反馈。如果您想就如何改善 DB2 文档提出建议，请向 db2docs@ca.ibm.com 发送电子邮件。DB2 文档小组将阅读您的所有反馈，但无法直接给您答复。请尽可能提供具体的示例，这样我们才能更好地了解您所关心的问题。如果您要提供有关具体主题或帮助文件的反馈，请加上标题和 URL。

请不要使用以上电子邮件地址与 DB2 客户支持机构联系。如果您遇到文档无法解决的 DB2 技术问题，请与您当地的 IBM 服务中心联系以获得帮助。

硬拷贝或 PDF 格式的 DB2 技术库

下列各表描述 IBM 出版物中心（网址为 www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss）所提供的 DB2 资料库。可从 www.ibm.com/support/docview.wss?rs=71&uid=swg2700947 下载 PDF 格式的 DB2 V10.1 手册的英文版本和翻译版本。

尽管这些表标识书籍有印刷版，但可能未在您所在国家或地区提供。

每次更新手册时，表单号都会递增。确保您正在阅读下面列示的手册的最新版本。

注: *DB2 信息中心*的更新频率比 PDF 或硬拷贝书籍的更新频率高。

表 43. DB2 技术信息

书名	书号	是否提供印刷版	最近一次更新时间
Administrative API Reference	SC27-3864-00	是	2012 年 4 月
Administrative Routines and Views	SC27-3865-01	否	2013 年 1 月
Call Level Interface Guide and Reference Volume 1	SC27-3866-01	是	2013 年 1 月
Call Level Interface Guide and Reference Volume 2	SC27-3867-01	是	2013 年 1 月
Command Reference	SC27-3868-01	是	2013 年 1 月
数据库管理概念和配置参考	S151-1758-01	是	2013 年 1 月
数据移动实用程序指南和参考	S151-1756-01	是	2013 年 1 月
数据库监视指南和参考	S151-1759-01	是	2013 年 1 月
数据恢复及高可用性指南与参考	S151-1755-01	是	2013 年 1 月
数据库安全性指南	S151-1753-02	是	2013 年 1 月
DB2 Workload Management Guide and Reference	SC27-3891-01	是	2013 年 1 月
开发 ADO.NET 和 OLE DB 应用程序	S151-1765-01	是	2013 年 1 月
开发嵌入式 SQL 应用程序	S151-1763-01	是	2013 年 1 月
Developing Java Applications	SC27-3875-01	是	2013 年 1 月
Developing Perl, PHP, Python, and Ruby on Rails Applications	SC27-3876-00	否	2012 年 4 月
Developing RDF Applications for IBM Data Servers	SC27-4462-00	是	2013 年 1 月
开发用户定义的例程（SQL 和外部例程）	S151-1761-01	是	2013 年 1 月
数据库应用程序开发入门	G151-1764-01	是	2013 年 1 月
Linux 和 Windows 上的 DB2 安装和管理入门	G151-1769-00	是	2012 年 4 月
全球化指南	S151-1757-00	是	2012 年 4 月
安装 DB2 服务器	G151-1768-01	是	2013 年 1 月
安装 IBM Data Server Client	G151-1751-00	否	2012 年 4 月
消息参考第 1 卷	S151-1767-01	否	2013 年 1 月

表 43. DB2 技术信息 (续)

书名	书号	是否提供印刷版	最近一次更新时间
消息参考第 2 卷	S151-1766-01	否	2013 年 1 月
Net Search Extender 管理和用户指南	S151-1905-01	否	2013 年 1 月
分区和集群指南	S151-1754-01	是	2013 年 1 月
Preparation Guide for DB2 10.1 Fundamentals Exam 610	SC27-4540-00	否	2013 年 1 月
Preparation Guide for DB2 10.1 DBA for Linux, UNIX, and Windows Exam 611	SC27-4541-00	否	2013 年 1 月
pureXML 指南	S151-1775-01	是	2013 年 1 月
Spatial Extender User's Guide and Reference	SC27-3894-00	否	2012 年 4 月
SQL 过程语言: 应用程序启用和支持	S151-1762-01	是	2013 年 1 月
SQL Reference Volume 1	SC27-3885-01	是	2013 年 1 月
SQL Reference Volume 2	SC27-3886-01	是	2013 年 1 月
Text Search Guide	SC27-3888-01	是	2013 年 1 月
故障诊断和调整数据库性能	S151-1760-01	是	2013 年 1 月
升级到 DB2 V10.1	S151-1770-01	是	2013 年 1 月
DB2 V10.1 新增内容	S151-1752-01	是	2013 年 1 月
XQuery 参考	S151-1774-01	否	2013 年 1 月

表 44. 特定于 DB2 Connect 的技术信息

书名	书号	是否提供印刷版	最近一次更新时间
DB2 Connect 安装和配置 DB2 Connect Personal Edition	S151-1773-00	是	2012 年 4 月
DB2 Connect 安装和配置 DB2 Connect 服务器	S151-1772-01	是	2013 年 1 月
DB2 Connect 用户指南	S151-1771-01	是	2013 年 1 月

从命令行处理器显示 SQL 状态帮助

DB2 产品针对可能充当 SQL 语句结果的条件返回 SQLSTATE 值。SQLSTATE 帮助说明 SQL 状态和 SQL 状态类代码的含义。

过程

要启动 SQL 状态帮助，请打开命令行处理器并输入:

```
? sqlstate or ? class code
```

其中，*sqlstate* 表示有效的 5 位 SQL 状态，*class code* 表示该 SQL 状态的前 2 位。例如，`? 08003` 显示 08003 SQL 状态的帮助，而 `? 08` 显示 08 类代码的帮助。

访问不同版本的 DB2 信息中心

您可以在 ibm.com® 上的不同信息中心中找到其他版本 DB2 产品的文档。

关于此任务

对于 DB2 V10.1 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v10r1。

对于 DB2 V9.8 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/。

对于 DB2 V9.7 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/。

对于 DB2 V9.5 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v9r5。

对于 DB2 V9.1 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v9/。

对于 DB2 V8 主题，请转至 *DB2 信息中心* URL: http://publib.boulder.ibm.com/infocenter/db2luw/v8/。

更新安装在计算机或内部网服务器上的 DB2 信息中心

安装在本地的 DB2 信息中心必须定期进行更新。

开始之前

必须已安装 DB2 V10.1 信息中心。有关详细信息，请参阅安装 DB2 服务器中的“使用 DB2 安装向导来安装 DB2 信息中心”主题。所有适用于安装信息中心的先决条件和限制同样适用于更新信息中心。

关于此任务

可以自动或手动更新现有的 DB2 信息中心:

- 自动更新将更新现有的信息中心功能部件和语言。自动更新的一个优点是，与手动更新相比，信息中心的不可用时间较短。另外，自动更新可设置为作为定期运行的其他批处理作业的一部分运行。
- 可以使用手动更新方法来更新现有的信息中心功能部件和语言。自动更新可以缩短更新过程中的停机时间，但如果您想添加功能部件或语言，那么必须执行手动过程。例如，如果本地信息中心最初安装的是英语和法语版，而现在还要安装德语版；那么手动更新将安装德语版，并更新现有信息中心的功能和语言。但是，手动更新要求您手动停止、更新和重新启动信息中心。在整个更新过程期间信息中心不可用。在自动更新过程中，信息中心仅在更新完成后停止工作以重新启动信息中心。

此主题详细说明了自动更新的过程。有关手动更新的指示信息，请参阅“手动更新安装在您的计算机或内部网服务器上的 DB2 信息中心”主题。

过程

要自动更新安装在计算机或内部网服务器上的 DB2 信息中心:

1. 在 Linux 操作系统上,
   a. 浏览至信息中心的安装位置。缺省情况下，DB2 信息中心安装在 /opt/ibm/db2ic/V10.1 目录中。
   b. 从安装目录浏览至 doc/bin 目录。
   c. 运行 update-ic 脚本:

      ```
      update-ic
      ```
2. 在 Windows 操作系统上,
   a. 打开命令窗口。
   b. 浏览至信息中心的安装位置。缺省情况下，DB2 信息中心安装在 <Program Files>\IBM\DB2 Information Center\V10.1 目录中，其中 <Program Files> 表示 Program Files 目录的位置。
   c. 从安装目录浏览至 doc\bin 目录。
   d. 运行 update-ic.bat 文件:

      ```
      update-ic.bat
      ```

结果

DB2 信息中心将自动重新启动。如果更新可用，那么信息中心会显示新的以及更新后的主题。如果信息中心更新不可用，那么会在日志中添加消息。日志文件位于 doc\eclipse\configuration 目录中。日志文件名称是随机生成的编号。例如，1239053440785.log。

手动更新安装在计算机或内部网服务器上的 DB2 信息中心

如果您已在本地安装 DB2 信息中心，那么可从 IBM 获取文档更新并进行安装。

关于此任务

手动更新安装在本地的 *DB2 信息中心*要求您:

1. 停止计算机上的 *DB2 信息中心*，然后以独立方式重新启动信息中心。如果以独立方式运行信息中心，那么网络上的其他用户将无法访问信息中心，因而您可以应用更新。DB2 信息中心的工作站版本总是以独立方式运行。
2. 使用“更新”功能部件来查看可用的更新。如果有您必须安装的更新，那么请使用“更新”功能部件来获取并安装这些更新。

   注: 如果您的环境要求在一台未连接至因特网的机器上安装 *DB2 信息中心*更新，请使用一台已连接至因特网并已安装 *DB2 信息中心*的机器将更新站点镜像至本地文件系统。如果网络中有许多用户将安装文档更新，那么可以通过在本地也为更新站点制作镜像并为更新站点创建代理来缩短每个人执行更新所需要的时间。
   如果提供了更新包，请使用“更新”功能部件来获取这些更新包。但是，只有在单机方式下才能使用“更新”功能部件。

3. 停止独立信息中心，然后在计算机上重新启动 *DB2 信息中心*。

注: 在 Windows 2008、Windows Vista 和更高版本上，稍后列示在此部分的命令必须作为管理员运行。要打开具有全面管理员特权的命令提示符或图形工具，请右键单击快捷方式，然后选择**以管理员身份运行**。

过程

要更新安装在您的计算机或内部网服务器上的 *DB2 信息中心*:

1. 停止 *DB2 信息中心*。
   - 在 Windows 上，单击**开始 > 控制面板 > 管理工具 > 服务**。右键单击 **DB2 信息中心**服务，并选择**停止**。
   - 在 Linux 上，输入以下命令:

     ```
     /etc/init.d/db2icdv10 stop
     ```
2. 以独立方式启动信息中心。
   - 在 Windows 上:
     a. 打开命令窗口。
     b. 浏览至信息中心的安装位置。缺省情况下，*DB2 信息中心*安装在 *Program_Files*\IBM\DB2 Information Center\V10.1 目录中，其中 *Program Files* 表示 Program Files 目录的位置。
     c. 从安装目录浏览至 doc\bin 目录。
     d. 运行 help_start.bat 文件:

        ```
        help_start.bat
        ```
   - 在 Linux 上:
     a. 浏览至信息中心的安装位置。缺省情况下，*DB2 信息中心*安装在 /opt/ibm/db2ic/V10.1 目录中。
     b. 从安装目录浏览至 doc/bin 目录。
     c. 运行 help_start 脚本:

        ```
        help_start
        ```

   系统缺省 Web 浏览器将打开以显示独立信息中心。
3. 单击**更新**按钮（  ）。（必须在浏览器中启用 JavaScript。）在信息中心的右边面板上，单击**查找更新**。将显示现有文档的更新列表。
4. 要启动安装过程，请检查您要安装的选项，然后单击**安装更新**。
5. 在安装进程完成后，请单击**完成**。
6. 要停止独立信息中心，请执行下列操作:
   - 在 Windows 上，浏览至安装目录中的 doc\bin 目录并运行 help_end.bat 文件:

     ```
     help_end.bat
     ```

     注: help_end 批处理文件包含安全地停止使用 help_start 批处理文件启动的进程所需的命令。不要使用 Ctrl-C 或任何其他方法来停止 help_start.bat。
   - 在 Linux 上，浏览至安装目录中的 doc/bin 目录并运行 help_end 脚本:

     ```
     help_end
     ```

注: help_end 脚本包含安全地停止使用 help_start 脚本启动的进程所需的命令。不要使用任何其他方法来停止 help_start 脚本。

7. 重新启动 *DB2 信息中心*。
   - 在 Windows 上，单击**开始 > 控制面板 > 管理工具 > 服务**。右键单击 **DB2 信息中心**服务，并选择**启动**。
   - 在 Linux 上，输入以下命令:

     ```
     /etc/init.d/db2icdv10 start
     ```

结果

更新后的 *DB2 信息中心*将显示新的以及更新后的主题。

DB2 教程

DB2 教程帮助您了解 DB2 数据库产品的各个方面。这些课程提供了逐步指示信息。

开始之前

您可以在信息中心中查看 XHTML 版的教程: http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/。

某些课程使用了样本数据或代码。有关其特定任务的任何先决条件的描述，请参阅教程。

DB2 教程

要查看教程，请单击标题。

*pureXML 指南*中的『**pureXML®**』
: 设置 DB2 数据库以存储 XML 数据以及对本机 XML 数据存储器执行基本操作。

DB2 故障诊断信息

我们提供了各种各样的故障诊断和问题确定信息来帮助您使用 DB2 数据库产品。

DB2 文档
: 您可以在《*故障诊断和调整数据库性能*》或者 *DB2 信息中心*的“数据库基础”部分中找到故障诊断信息，这些信息包含以下内容:
  - 有关如何使用 DB2 诊断工具和实用程序来隔离和确定问题的信息。
  - 一些最常见问题的解决方案。
  - 旨在帮助您解决 DB2 数据库产品使用过程中可能会遇到的其他问题的建议。

IBM Support Portal
: 如果您遇到问题并且希望得到帮助以查找可能的原因和解决方案，请访问 IBM Support Portal 。这个技术支持站点提供了指向最新 DB2 出版物、技术说明、授权程序分析报告（APAR 或错误修订）、修订包和其他资源的链接。可搜索此知识库并查找问题的可能解决方案。

  访问 IBM 支持门户网站网址为: http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows

信息中心条款和条件

如果符合以下条款和条件，那么授予您使用这些出版物的许可权。

适用性: 用户需要遵循 IBM Web 站点的使用条款及以下条款和条件。

个人使用: 只要保留所有的专有权声明，您就可以为个人、非商业使用复制这些出版物。未经 IBM 明确同意，您不可以分发、展示或制作这些出版物或其中任何部分的演绎作品。

商业使用: 只要保留所有的专有权声明，您就可以仅在企业内复制、分发和展示这些出版物。未经 IBM 明确同意，您不可以制作这些出版物的演绎作品，或者在您的企业外部复制、分发或展示这些出版物或其中的任何部分。

权利: 除非本许可权中明确授予，否则不得授予对这些出版物或其中包含的任何信息、数据、软件或其他知识产权的任何许可权、许可证或权利，无论是明示的还是暗含的。

IBM 保留根据自身的判断，认为对出版物的使用损害了 IBM 的权益（由 IBM 自身确定）或未正确遵循以上指示信息时，撤回此处所授予权限的权利。

只有您完全遵循所有适用的法律和法规，包括所有的美国出口法律和法规，您才可以下载、出口或再出口该信息。

IBM 对这些出版物的内容不作任何保证。这些出版物“按现状”提供，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的关于适销和适用于某种特定用途的保证。

IBM® 商标: IBM、 IBM 徽标和 ibm.com 是 International Business Machines Corp.，在全球许多管辖区域注册的商标或注册商标。其他产品和服务名称可能是 IBM 或其他公司的商标。当前的 IBM 商标列表，可从 Web 站点 www.ibm.com/legal/copytrade.shtml获得

附录 B. 声明

本信息是为在美国提供的产品和服务编写的。有关非 IBM 产品的信息是基于首次出版此文档时的可获信息且会随时更新。

IBM 可能在其他国家或地区不提供本文档中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可。您可以用书面方式将许可查询寄往:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节字符集 (DBCS) 信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区: International Business Machines Corporation“按现状”提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某种特定用途的保证。某些国家或地区在某些交易中不允许免除明示或暗含的保证。因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本资料的新版本中。IBM 可以随时对本资料中描述的产品和/或程序进行改进和/或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是此 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的: (i) 允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及 (ii) 允许对已经交换的信息进行相互使用，请与下列地址联系:

IBM Canada Limited
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

只要遵守适当的条款和条件，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际软件许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其他操作环境中获得的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

本信息可能包含在日常业务操作中使用的数据和报告的示例。为了尽可能完整地说明这些示例，示例中可能会包括个人、公司、品牌和产品的名称。所有这些名称都是虚构的，与实际商业企业所用的名称和地址的任何雷同纯属巧合。

版权许可:

本信息包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。如果是为按照在编写样本程序的操作平台上的应用程序编程接口 (API) 进行应用程序的开发、使用、经销或分发，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例并未在所有条件下作全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。此样本程序“按现状”提供，且不附有任何种类的保证。对于使用此样本程序所引起的任何损坏，IBM 将不承担责任。

凡这些样本程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明:

© (*your company name*) (*year*). 此部分代码是根据 IBM 公司的样本程序衍生出来的。
© Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

商标

IBM 商标: IBM、 IBM 徽标和 ibm.com 是 International Business Machines Corp., 在全球许多管辖区域注册的商标或注册商标。其他产品和服务名称可能是 IBM 或其他公

司的商标。当前的 IBM 商标列表，可从 Web 站点 www.ibm.com/legal/copytrade.shtml 上“版权和商标信息”部分获取。

下列各项是其他公司的商标或注册商标

- Linux 是 Linus Torvalds 在美国和/或其他国家或地区的注册商标。
- Java™ 和所有基于 Java 的商标和徽标是 Oracle 和/或其子公司的商标或注册商标。
- UNIX 是 The Open Group 在美国和其他国家或地区的注册商标。
- Intel、Intel 徽标、Intel Inside、Intel Inside 徽标、Celeron、Intel SpeedStep、Itanium 和 Pentium 是 Intel Corporation 或其子公司在美国和其他国家或地区的商标或注册商标。
- Microsoft、Windows、Windows NT 和 Windows 徽标是 Microsoft Corporation 在美国和/或其他国家或地区的商标。

其他公司、产品或服务名称可能是其他公司的商标或服务标记。

索引

[B]

[J]

[K]

[L]

[M]

[N]

[P]

[Q]

[R]

[S]

[T]

[W]

[X]

[Y]

[Z]

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

W

X

Y

Z

Printed in China

S151-1774-01

Spine information:

IBM DB2 10.1 for Linux, UNIX, and Windows

XQuery 参考