变量(Variables)

变量（Variables）

QL中的变量与代数或逻辑中的变量的使用方式相似。它们表示一组值，这些值通常受公式限制。
这与某些其他编程语言中的变量不同，在其他编程语言中，变量表示可能包含数据的内存位置。该数据还可以随着时间变化。例如，在QL中，`n = n + 1`是一个等式公式，仅当`n`等于`n + 1`时才成立（因此，实际上它不适用于任何数值）。在Java中，`n = n + 1`不是等式，而是通过添加`1`到当前的值`n`，并进行新的赋值来改变`n`代表的值。

声明一个变量（Declaring a variable）

所有变量声明均由变量的类型和名称组成。名称可以是任何 以大写或小写字母开头的标识符。
例如，int i、 `LocalScopeVariable` node 和 `LocalScopeVariable` lsv 声明变量 i、 node 和 lsv，它们的类型分别为 int、 SsaDefinitionNode和 `LocalScopeVariable` 。
变量声明出现在不同的上下文中，例如在select子句，量化公式内，作为谓词的参数等。
从概念上讲，您可以将变量视为保留其类型允许的所有值，但要受到其他约束。
例如，考虑以下select子句：

```
from int i
where i in [0 .. 9]
select i
```

仅根据其类型，变量`i`可以包含所有整数。但是，它受公式`i in [0 .. 9]`约束。因此，select子句的结果是介于`0`到`9`之间的数字。
顺便说一句，请注意以下查询会导致编译时错误：

```
from int i
select i
```

从理论上讲，它将得到无穷多个结果，因为该变量不限`i`于有限数量的可能值

自由变量和约束变量(Free and bound variables)

变量可以具有不同的作用。有些变量是**free**，它们的值直接影响使用它们的表达式的值，或者使用它们的公式是否 成立。其他变量（称为**绑定**变量）仅限于特定的值集。
在一个示例中，最容易理解这种区别。看一下以下表达式：

```
"hello".indexOf("l")

min(float f | f in [-3 .. 3])

(i + 7) * 3

x.sqrt()
```

第一个表达式没有任何变量。它找到`"l"`字符串中出现位置的（从零开始的）索引`"hello"`，因此它的结果为`2`和`3`。
第二个表达式的计算结果为`[-3 .. 3]`范围内的最小值`-3`。尽管此表达式使用变量`f`，但它只是一个占位符或“虚拟”变量，您不能为其分配任何值。您可以使用其他变量替换`f`而无需更改表达式的含义。例如，`min(float f | f in [-3 .. 3])`始终等于`min(float other | other in [-3 .. 3])` 。这是**绑定变量**的示例。
表达式`(i + 7) * 3`和`x.sqrt()`有该是怎样的呢？在这两种情况下，表达式的值取决于什么值赋值给变量和分别。换句话说，变量的值对表达式的值有影响。这些是**自由变量的**示例。
同样，如果一个公式包含自由变量，则该公式可以保留还是不保留，取决于分配给这些变量的值。例如：

```
"hello".indexOf("l") = 1

min(float f | f in [-3 .. 3]) = -3

(i + 7) * 3 instanceof int

exists(float y | x.sqrt() = y)
```

第一个公式不包含任何变量，并且永远不成立（因为它`"hello".indexOf("l")` 具有值`2`和`3`，从不`1`）。
第二个公式仅包含一个绑定变量，因此不受该变量更改的影响。由于等于，所以该公式始终成立。
`min(float f | f in [-3 .. 3])` `-3`

第三个公式包含一个自由变量`i`。公式是否成立，取决于分配给的值`i`。例如，如果`i`分配了`1`或`2`（或任何其他`int`），则公式成立。另一方面，如果`i`已分配`3.5`，则它不成立。
最后一个公式包含一个自由变量`x`和一个绑定变量`y`。如果`x`分配了一个非负数，则最终公式成立。另一方面，例如，如果`x`指定 `-9`，则公式不成立。该变量`y`不影响公式是否成立。