谓词(Predicates)

谓词(Predicates)

谓词用于描述组成QL程序的逻辑关系。
严格来说，谓词对一组元组求值。例如，考虑以下两个谓词定义：

```
predicate isCountry(string country) {
  country = "Germany"
  or
  country = "Belgium"
  or
  country = "France"
}

predicate hasCapital(string country, string capital) {
  country = "Belgium" and capital = "Brussels"
  or
  country = "Germany" and capital = "Berlin"
  or
  country = "France" and capital = "Paris"
}
```

谓词`isCountry`是一元组的集合`{("Belgium"),("Germany"),("France")}`，而谓词是二元组`hasCapital`的集合`{("Belgium","Brussels"),("Germany","Berlin"),("France","Paris")}`。这些谓词的Arity分别为1和2。

通常，谓词中的所有元组都具有相同数量的元素。谓词的**多样性**是不包含可能的`result`变量的元素数量。有关更多信息，请参见“带有结果的谓词”。

QL中有许多内置谓词。您可以在任何查询中使用它们，而无需导入 任何其他模块。除了这些内置谓词，您还可以定义自己的谓词

定义谓词(Defining a predicate)

定义谓词时，应指定：

- 关键字predicate（对于没有结果的谓词），或者结果的类型（对于具有结果的谓词）
- 谓词的名称。这是一个 以小写字母开头的标识符
- 谓词的参数（如果有）以逗号分隔。对于每个参数，请指定参数类型和参数变量的标识符
- 谓词体本身。这是用大括号括起来的逻辑公式

```
笔记
一个抽象的或外部的谓词有没有实体。要定义这样的谓词，请以分号（;）结尾。
```

没有结果的谓词(Predicates without result)

这些谓词定义以关键字开头`predicate`。如果值满足主体中的逻辑属性，则谓词将保留该值。

例如：

```
predicate isSmall(int i) {
  i in [1 .. 9]
}
```

如果`i`是整数，则`isSmall(i)`保存如果`i`是小于10的正整数。

有结果的谓词(Predicates with result)

您可以通过将关键字替换为结果`predicate`的类型来定义带有结果的谓词。这引入了特殊变量`result`。

例如：

```
int getSuccessor(int i) {
  result = i + 1 and
```

```
  i in [1 .. 9]
}
```

如果`i`是小于10的正整数，则谓词的结果为的后继`i`
请注意，您可以使用`result`与谓词的任何其他参数相同的方式。您可以`result`按照自己喜欢的任何方式来表达与其他变量之间的关系。例如，给定一个返回的父代的谓词，您可以如下定义“反向”谓词：`getAParentOf(Person x)` `x`

```
Person getAChildOf(Person p) {
  p = getAParentOf(result)
}
```

对于谓词的每个值，谓词也可能具有多个结果（或根本没有）。例如：

```
string getANeighbor(string country) {
  country = "France" and result = "Belgium"
  or
  country = "France" and result = "Germany"
  or
  country = "Germany" and result = "Austria"
  or
  country = "Germany" and result = "Belgium"
}
```

在这种情况下：

- 谓词调用`getANeighbor("Germany")`返回两个结果：`"Austria"`和 `"Belgium"`。
- 谓词调用`getANeighbor("Belgium")`不返回任何结果，因为`getANeighbor` 未定义`result`for `"Belgium"`。

递归谓词（Recursive predicates）

QL中的谓词可以是**递归的**。这意味着它直接或间接取决于自身。
例如，您可以使用递归来完善上面的示例。就目前而言，`getANeighbor`中定义的关系不是对称的–它无法捕获以下事实：如果x是y的邻居，则y是x的邻居。一种简单的捕获此谓词的方法是递归调用此谓词，如下所示：

```
string getANeighbor(string country) {
  country = "France" and result = "Belgium"
```

```
  or
  country = "France" and result = "Germany"
  or
  country = "Germany" and result = "Austria"
  or
  country = "Germany" and result = "Belgium"
  or
  country = getANeighbor(result)
}
```

现在`getANeighbor("Belgium")`还返回结果，即`"France"`和`"Germany"`。
有关递归谓词和查询的更一般性讨论，请参见“递归”。

谓词的种类（Kinds of predicates）

存在三种谓词，即非成员谓词，成员谓词和特征谓词。
非成员谓词是在类之外定义的，也就是说，它们不是任何类的成员。
有关其他谓词的更多信息，请参见“类”主题中的特征谓词和成员谓词。
这是显示每种谓词的示例：

```
int getSuccessor(int i) {  // 1. Non-member predicate
  result = i + 1 and
  i in [1 .. 9]
}

class FavoriteNumbers extends int {
  FavoriteNumbers() {  // 2. Characteristic predicate
    this = 1 or
    this = 4 or
    this = 9
  }

  string getName() {  // 3. Member predicate for the class `Favo
riteNumbers`
    this = 1 and result = "one"
    or
    this = 4 and result = "four"
    or
    this = 9 and result = "nine"
  }
```

```
20 }
```

您还可以注释每个谓词。请参阅可用于每种谓词的注释列表 。

绑定行为（Binding behavior）

必须有可能在有限的时间内评估谓词，因此通常不允许它描述的集合是无限的。换句话说，谓词只能包含有限数量的元组。

当QL编译器可以证明谓词包含的变量不限于有限数量的值时，将报告错误。有关更多信息，请参见“绑定”。

以下是无限谓词的一些示例：

```
1 /*
2   Compilation errors:
3   ERROR: "i" is not bound to a value.
4   ERROR: "result" is not bound to a value.
5   ERROR: expression "i * 4" is not bound to a value.
6 */
7 int multiplyBy4(int i) {
8   result = i * 4
9 }
10
11 /*
12   Compilation errors:
13   ERROR: "str" is not bound to a value.
14   ERROR: expression "str.length()" is not bound to a value.
15 */
16 predicate shortString(string str) {
17   str.length() < 10
18 }
```

在中`multiplyBy4`，参数`i`声明为`int`，这是一个无限类型。它用在二进制操作中`*`，该操作不绑定其操作数。 `result`首先是未绑定的，并且由于与相等的检查中使用了它，因此也保持未绑定的状态。`i * 4`

中的`shortString`， `str`由于使用无限类型声明，因此保持未绑定状态 `string`，并且内置函数`length()`不会绑定它。

绑定设置（Binding Set）

有时您可能仍然想定义一个“无限谓词”，因为您只打算在一组受限的参数上使用它。在这种情况下，您可以使用`bindingset` 批注指定显式绑定集。该注释对任何种类的谓词均有效。
例如：

```
bindingset[i]
int multiplyBy4(int i) {
  result = i * 4
}

from int i
where i in [1 .. 10]
select multiplyBy4(i)
```

尽管`multiplyBy4`是一个无限谓词，但上述QL查询是合法的。它首先使用`bindingset`注释声明谓词`multiplyBy4`将是有限的，前提`i`是该谓词绑定到有限数量的值。然后，在`i`限定于范围的上下文中使用谓词。`[1 .. 10]`
也可以为一个谓词声明多个绑定集。这可以通过添加多个绑定集注释来完成，例如：

```
bindingset[x] bindingset[y]
predicate plusOne(int x, int y) {
  x + 1 = y
}

from int x, int y
where y = 42 and plusOne(x, y)
select x, y
```

以这种方式指定的多个绑定集彼此独立。上面的示例表示：

- 如果`x`绑定，则`x`和`y`绑定。
- 如果`y`绑定，则`x`和`y`绑定。

也就是说，指出必须或必须绑定的至少一个不同于，指出必须和必须绑定的。`bindingset[x] bindingset[y]` `x` `y` `bindingset[x, y]` `x` `y`
当您要声明带有多个输入参数的结果的谓词时，后者很有用。例如，以下谓词采用字符串`str`并将其截断为最大`len`字符长度：

```
bindingset[str, len]
string truncate(string str, int len) {
```

```
  if str.length() > len
  then result = str.prefix(len)
  else result = str
}
```

然后可以在select子句中使用它，例如：

```
select truncate("hello world", 5)
```

数据库谓词(Database predicates)

您查询的每个数据库都包含表示值之间关系的表。这些表（“数据库谓词”）的处理方式与QL中的其他谓词相同。

例如，如果一个数据库包含人的表，你可以写 约束，以及将在该表行的第一，第二和第四列。

`persons(x, firstName, _, age)` `x` `firstName` `age`

唯一的区别是您不能在QL中定义数据库谓词。它们由基础数据库定义。因此，可用的数据库谓词根据要查询的数据库而有所不同。

小结

本节我们学习了谓词的定义以及一些定义谓词的方式，简单的说谓词就是将一些数据用一定的限定起来的集合规则。