Язык M Power Query. Структура управления

В названии нет опечатки. В мире Power Query нет управляющих структур (множественное число). Есть только одна управляющая структура. Мы изучим её. Что касается «отсутствующих» структур управления (к которым вы, привыкли в других языках программирования), мы рассмотрим несколько примеров обходных способов реализации аналогичной функциональности в коде M.[1]

Предыдущая заметка Следующая заметка

Условные выражения if

if в Power Query позволяет выполнить одно из двух выражений по условию:

```
if TestExpression
then ExpressionToUseWhenTestEvaluatesToTrue
else ExpressionToUseWhenTestEvaluatesToFalse
```

Сначала вычисляется значение условия – *TestExpression*. Если результат – *true*, вычисляется выражение *then* и возвращается его значение. Если результат – *false*, вычисляется выражение *else*. Если *TestExpression* не логическое значение, возвращается ошибка.

Допустим, у нас есть результаты тестов. Необходимо добавить столбец, указывающий, является ли результат проходным или нет. На помощь приходит условное выражение:

```
let
  Source = GetTestScoresFromSource(),
  EvaluateScore = (score) => if score >= 0.7 then "Pass" else "Fail",
  Result = Table.AddColumn(Source, "Result", each EvaluateScore([Score]))
in
  Result
```

Некоторые языки программирования допускают выражение *if* без части *else*. Это не разрешено в Power Query. В M каждое выражение (включая выражения *if*) должно либо возвращать значение, либо вызывать ошибку. Использование *if* без *else* приведет к тому, что ничего не произойдет, когда тест *if* получит значение *false*. Это нарушит работу M.

В отличие от других языков, в M нет краткого варианта синтаксиса для *if*. Поэтому не работает что-то типа:

```
score >= 0.85, "Pass", "Fail"
```

Правда, краткий синтаксис всё же встречается. Предположим, вы хотите вернуть само значение, если оно не равно *null*. А если оно равно *null* – альтернативное значение. Вы могли бы написать…

```
if ValueA <> null then ValueA else ValueB
```

… но для этого выражения есть краткая форма, использующая *оператор объединения с null* (??)

```
ValueA ?? ValueB
```

Листинг 1[2]

```
let
   a = null,
   b = 6,
   Result = a ?? b
in
   Result // возвращает 6
```

В M нет *elseif*. Чтобы применить несколько условий, вложите еще одну конструкцию *if* в часть *else*.

```
(Grade) =>
if Grade= "A" then 0.9
else if Grade= "B" then 0.8
```

---

[1] Заметка написана на основе статьи Ben Gribaudo. Power Query M Primer (Part 14): Control Structure. Если вы впервые сталкиваетесь с Power Query, рекомендую начать с Марк Мур. Power Query.

[2] Номер листинга соответствует номеру запроса в приложенном Excel файле.

```
else if Grade= "C" then 0.7
else if Grade= "D" then 0.6
else 0
```

Использование большого числа вложенных конструкций *if* нарушает принцип «Не повторяйся» (Don’t repeat yourself). К сожалению, в M нет аналогов конструкции switch. Но можно рассмотреть набор условных выражений как значений поиска, помещая их в таблицу а затем извлекая нужное.

```
(Grade) =>
let
   Map = #table(
      {"LetterGrade", "Score"},
      {
         {"A", 0.9},
         {"B", 0.8},
         {"C", 0.7},
         {"D", 0.6},
         {"F", 0}
      }
   )
in
   Map{[LetterGrade = Grade]}[Score]
```

Вместо таблицы выбор можно «завернуть» в запись:

```
(Grade) => Record.Field([A = 0.9, B = 0.8, C = 0.7, D = 0.6, F = 0], Grade)
```

Или можно вернуть 0, как значение по умолчанию, если оценка не соответствует ни одному имени поля записи:

```
(Grade) => Record.FieldOrDefault([A = 0.9, B = 0.8, C = 0.7, D = 0.6], Grade, 0)
```

Циклы

В языке M нет иных структур управления. Не поддерживаются циклы типа *foreach*, *for* или *while/do while*. Вы можете подумать, что M – незрелый язык, в котором отсутствуют столь важные понятия. Не так быстро! Функциональное программирование в целом и M в частности отличаются от того, к чему вы привыкли. Функциональные языки полагаются на иные средства для достижения того же результата. Давайте рассмотрим эти подходы.

Конструкция foreach

Традиционно выражение *foreach* выполняет блок кода для каждого элемента в коллекции (например, для каждого элемента в списке или каждой строки в таблице). Power Query изначально был создан для обработки элементов списка и строк таблицы. Поэтому вместо явного кодирования низкоуровневого цикла вы декларативно используете библиотечные функции. Они на более высоком уровне определяют, что вы хотите выполнить. Вся низкоуровневая обработка зашита в функции.

Например, чтобы применить преобразование к каждому значению в столбце, вы используете *Table.TransformColumns.* Функция применяет преобразование к указанному столбцу, а «за кулисами» M заботится о построчном применении этого преобразования. Аналогично, чтобы добавить новый столбец, вы не перебираете строки таблицы, добавляя значения нового столбца в каждую строку по одному за раз. Вместо этого вы просто заявляете о своем намерении с помощью *Table.AddColumn*, предоставляя этому методу выражение, определяющее новый столбец, а M заботится о расчете для каждой строки.

Поскольку ваше внимание сосредоточено на объявлении намерения, а не на кодировании обработки по строкам, результирующий синтаксис может оказаться более кратким. Например, вместо того, чтобы императивно использовать *foreach* для суммирования значений в столбце, в мире M вы просто объявляете это намерение, применив *List.Sum* к столбцу.

```
// Вместо…
var total = 0;
```

```
foreach(var row in SomeTable) {
  total += row[ColumnOfInterest];
}
// … вы используете…
List.Sum(SomeTable[ColumnOfInterest])
```

for, while, do while

В ряде других языков программирования *for* используется для итерации определенное количество раз. Например, начать с 0; добавлять по одному за каждую итерацию; остановиться при достижении 10. *while* и *do while* выполняют цикл пока проверка не вернет *false*.

Если что-то подобное вам нужно в мире M, сначала создайте ряд значений, используя функции генерации списков *List.Numbers* или *List. Generate*. Затем к этому ряду примените требуемую обработку в стиле *выполнить для каждого элемента*. Чтобы остановить обработку в нужный момент используйте функцию *List.Select*.

Предположим, что вы ожидаете завершения длительного удаленного задания. У вас есть функция, которая проверяет статус задания, вызывая веб-службу. Возвращается значение null, если задание все еще выполняется, или результат задания, если оно завершено. Вы хотите повторно запрашивать источник до тех пор, пока либо не получите ненулевой ответ (указывающий на завершение), либо не попробуете определенное количество раз (ограничение безопасности). В императивном языке программирования вы могли бы использовать цикл *for* (или *do while*) с предложением *break*. В M вы можете сделать это, используя что-то вроде:

```
(MaxAttempts, DelayBetweenAttempts) =>
let
   Numbers = List.Numbers(1, MaxAttempts),
   WebServiceCalls = List.Transform(
      Numbers, each Function.InvokeAfter(
         CallWebService,
         if _ > 1 then DelayBetweenAttempts else #duration(0,0,0,0)
      )
   ),
   OnlySuccessful = List.Select(WebServiceCalls, each _ <> null),
   Result = List.First(OnlySuccessful, null)
in
   Result
```

Здесь во всей красота проявляется потоковая передача. Обработка начинается с шага *Result*. Функция *List.First* запрашивает шаг *OnlySuccessful* для одного элемента списка. На шаге *Onlysuccessful* функция *List.Select* извлекает по одному значению за раз из *WebServiceCalls*, пока не найдет значение, отличное от *null*, которое возвращается на шаг *Result*.

Каждый раз, когда значение извлекается из *WebServiceCalls*, функция *List.Transform* через *CallWebService* вызывает веб-службу. Предельное число запросов веб-службы задается при вызове функции в первом параметре – *MaxAttempts*.

Как только будет возвращено первое ненулевое значение, потоковая передача прекратится, и вызовы веб-службы также прекратятся. Это верно, даже если не будет достигнуто значение *MaxAttempts.* То есть, веб-служба запрашивается до тех пор, пока не вернет ненулевой результат, в пределах заданного количества попыток. Красиво!

Если вам нужно бесконечно повторять цикл до тех пор, пока не будет достигнуто желаемое состояние (по сравнению с фиксированным числом раз, как в примере выше), вы можете управлять итерацией, используя бесконечный ряд, созданный *List.Generate*:

```
List.Generate(() => null, each true, each null)
```

Если вам нужно повторять цикл много-много раз, скорее всего, либо существует неитеративный подход, который вы могли бы (должны) использовать, либо вы делаете что-то за пределами

сильной стороны Power Query (помимо извлечения и обработки данных). В последнем случае лучше использовать более подходящий язык / технологию.

Бонус. Реализация логики *ожидание-повторная попытка*, которая подключается к действию (например, вызов веб-службы) и проверяет успешность шага, в качестве аргументов функции *List.Generate* используйте *Value.WaitFor*, см. описание и определение.

Заключение

Итак, в M есть единственная управляющая структура – условная конструкция *if*. Мы также рассмотрели несколько обходных манеров, чтобы выполнить цикл или итерацию. Приведенные примеры не исчерпывают моделирование циклов в функциональном мире Power Query. Но они служат отправной точкой, которая поможет вам сориентироваться, когда вы столкнетесь с потребностью в цикле или повторении.

В следующий раз мы рассмотрим обработку ошибок в Power Query.