F#: шаблоны, типы. Базовые паттерны ФП.

Юрий Литвинов

04.03.2016г

Сопоставление шаблонов

F#

```
let urlFilter url agent =
    match (url, agent) with
    | "http://www.google.com", 99 -> true
    | "http://www.yandex.ru" , _ -> false
    | _, 86 -> true
    | _ -> false
```

F#

```
let sign x =
    match x with
    | _ when x < 0 -> -1
    | _ when x > 0 -> 1
    | _ -> 0
```

F# — не Prolog

Не получится писать так:

F#

```
let isSame pair =
    match pair with
    | (a, a) -> true
    | _ -> false
```

Нужно так:

F#

```
let isSame pair =
    match pair with
    | (a, b) when a = b -> true
    | _ -> false
```

Какие шаблоны бывают

Синтаксис	Описание	Пример
$(pat, \ldots, pat)$	Кортеж	$(1, 2, (\text{“}3\text{“}, x))$
$[pat; \ldots; pat]$	Список	$[x; y; 3]$
pat :: *pat*	cons	*h* :: *t*
pat | *pat*	”Или”	$[x] \mid [\text{“}X\text{“} \mid x]$
pat & *pat*	”И”	$[p]\&[(x, y)]$
pat as id	Именованный шаблон	$[x]$ *as inp*
id	Переменная	*x*
_	Wildcard (что угодно)	_
литерал	Константа	239, *DayOfWeek.Monday*
:? *type*	Проверка на тип	:? *string*

Последовательности

Ленивый тип данных

F#

```
seq {0 .. 2}
seq {1I .. 10000000000000I}
```

F#

```
open System.IO
let rec allFiles dir =
    Seq.append
    (dir |> Directory.GetFiles)
    (dir |> Directory.GetDirectories
         |> Seq.map allFiles
         |> Seq.concat)
```

Типичные операции с последовательностями

Операция	Тип
Seq.append	$\#seq <' a> \rightarrow \#seq <' a> \rightarrow seq <' a>$
Seq.concat	$\#seq < \#seq <' a >> \rightarrow seq <' a>$
Seq.choose	$('a \rightarrow' boption) \rightarrow \#seq <' a> \rightarrow seq <' b>$
Seq.empty	$seq <' a>$
Seq.map	$('a \rightarrow' b) \rightarrow \#seq <' a> \rightarrow \#seq <' b>$
Seq.filter	$('a \rightarrow bool) \rightarrow \#seq <' a> \rightarrow seq <' a>$
Seq.fold	$('s \rightarrow' a \rightarrow' s) \rightarrow' s \rightarrow seq <' a> \rightarrow' s$
Seq.initInfinite	$(int \rightarrow' a) \rightarrow seq <' a>$

Задание последовательностей

F#

```
let squares = seq { for i in 0 .. 10 -> (i, i * i) }
seq { for (i, isquared) in squares ->
          (i, isquared, i * isquared) }
```

F#

```
let checkerboardCoordinates n =
    seq { for row in 1 .. n do
          for col in 1 .. n do
              if (row + col) % 2 = 0 then
                  yield (row, col) }
```

Ленивое чтение из файла

F#

```
let rec allFiles dir =
    seq { for file in Directory.GetFiles(dir) -> file
          for subdir in Directory.GetDirectories dir ->>
              (allFiles subdir) }
```

F#

```
let reader =
    seq {
        use reader = new StreamReader(
            File.OpenRead("test.txt")
            )
        while not reader.EndOfStream do
            yield reader.ReadLine() }
```

Записи

F#

```
type Person =
    { Name: string;
      DateOfBirth: System.DateTime; }
```

F#

```
{ Name = "Bill";
  DateOfBirth = new System.DateTime(1962, 09, 02) }

{ new Person
  with Name = "Anna"
  and DateOfBirth = new System.DateTime(1968, 07, 23) }
```

Клонирование записей

F#

```
type Car =
    {
        Make : string
        Model : string
        Year : int
    }
```

F#

```
let thisYear's = { Make = "SomeCar";
                   Model = "Luxury Sedan";
                   Year = 2010 }
let nextYear's = { thisYear's with Year = 2011 }
```

Размеченные объединения

Discriminated unions

F#

```
type Route = int
type Make = string
type Model = string

type Transport =
    | Car of Make * Model
    | Bicycle
    | Bus of Route
```

Известные примеры

F#

```
type 'a option =
    | None
    | Some of 'a
```

F#

```
type 'a list =
    | ([])
    | (::) of 'a * 'a list
```

Использование размеченных объединений

F#

```
type IntOrBool = I of int | B of bool
let i = I 99
let b = B true
```

F#

```
type C = Circle of int | Rectangle of int * int

[1..10]
|> List.map Circle

[1..10]
|> List.zip [21..30]
|> List.map Rectangle
```

Использование в match

F#

```
type Tree<'a> =
    | Tree of 'a * Tree<'a> * Tree<'a>
    | Tip of 'a

let rec size tree =
    match tree with
    | Tree(_, l, r) -> 1 + size l + size r
    | Tip _ -> 1
```

Пример

Дерево разбора логического выражения

F#

```
type Proposition =
    | True
    | And of Proposition * Proposition
    | Or of Proposition * Proposition
    | Not of Proposition

let rec eval (p: Proposition) =
    match p with
    | True -> true
    | And(p1, p2) -> eval p1 && eval p2
    | Or (p1, p2) -> eval p1 || eval p2
    | Not(p1) -> not (eval p1)

printfn "%A" <| eval (Or(True, And(True, Not True)))
```

Взаимосвязанные типы

F#

```
type node =
    { Name : string;
      Links : link list }
and link =
    | Dangling
    | Link of node
```

Замена цикла рекурсией

Императивное разложение на множители

F#

```
let factorizeImperative n =
    let mutable primefactor1 = 1
    let mutable primefactor2 = n
    let mutable i = 2
    let mutable fin = false
    while (i < n && not fin) do
        if (n % i = 0) then
            primefactor1 <- i
            primefactor2 <- n / i
            fin <- true
        i <- i + 1
    if (primefactor1 = 1) then None
    else Some (primefactor1, primefactor2)
```

Замена цикла рекурсией

Рекурсивное разложение на множители

F#

```
let factorizeRecursive n =
    let rec find i =
        if i >= n then None
        elif (n % i = 0) then Some(i, n / i)
        else find (i + 1)
    find 2
```

Хвостовая рекурсия, проблема

Императивный вариант

F#

```
open System.Collections.Generic

let createMutableList() =
    let l = new List<int>()
    for i = 0 to 100000 do
        l.Add(i)
    l
```

Хвостовая рекурсия, проблема

Рекурсивный вариант, казалось бы

F#

```
let createImmutableList() =
    let rec createList i max =
        if i = max then
            []
        else
            i :: createList (i + 1) max
    createList 0 100000
```

Факториал без хвостовой рекурсии

F#

```
let rec factorial x =
    if x <= 1
    then 1
    else x * factorial (x - 1)
```

F#

```
let rec factorial x =
    if x <= 1
    then
        1
    else
        let resultOfRecusion = factorial (x - 1)
        let result = x * resultOfRecusion
        result
```

Факториал с хвостовой рекурсией

F#

```
let factorial x =
    let rec tailRecursiveFactorial x acc =
        if x <= 1 then
            acc
        else
            tailRecursiveFactorial (x - 1) (acc * x)
    tailRecursiveFactorial x 1
```

После декомпиляции в C#

C#

```
public static int tailRecursiveFactorial(int x, int acc)
{
    while (true)
    {
        if (x <= 1)
        {
            return acc;
        }
        acc *= x;
        x--;
    }
}
```

Паттерн “Аккумулятор”

F#

```
let rec map f list =
    match list with
    | [] -> []
    | hd :: tl -> (f hd) :: (map f tl)

let map f list =
    let rec mapTR f list acc =
        match list with
        | [] -> acc
        | hd :: tl -> mapTR f tl (f hd :: acc)
    mapTR f (List.rev list) []
```

Аккумулятор — функция

F#

```
let printListRev list =
    let rec printListRevTR list cont =
        match list with
        | [] -> cont ()
        | hd :: tl ->
            printListRevTR tl (fun () ->
                printf "%d " hd; cont () )
    printListRevTR list (fun () -> printfn "Done!")
```

Когда всё не так просто

F#

```
type ContinuationStep<'a> =
    | Finished
    | Step of 'a * (unit -> ContinuationStep<'a>)

let rec linearize binTree cont =
    match binTree with
    | Empty -> cont()
    | Node(x, l, r) ->
        Step(x, (fun () -> linearize l (fun () ->
                                linearize r cont)))
```

Собственно, обход

F#

```
let iter f binTree =
    let steps = linearize binTree (fun () -> Finished)

    let rec processSteps step =
        match step with
        | Finished -> ()
        | Step(x, getNext) -> f x
            processSteps (getNext())

    processSteps steps
```