Введение в F#

Юрий Литвинов

22.02.2019г

F#

- Типизированный функциональный язык для платформы .NET
- НЕ чисто функциональный (можно императивный стиль и ООП)
- Первый раз представлен публике в 2005 г.
- Создавался под влиянием OCaml (практически диалект OCaml под .NET)
- Использует .NET CLI
- Компилируемый и интерпретируемый
- Используется в промышленности, в отличие от многих чисто функциональных языков

Что скачать и поставить

- Под Windows — Visual Studio, из коробки
- Под Linux
  - Rider (студентам бесплатно)
  - Mono + MonoDevelop + F# Language Binding, из репозиториев
  - .NET Core + Visual Studio Code + Ionide
- Прямо в браузере: https://dotnetfiddle.net/

Пример программы

```
printfn "%s" "Hello, world!"
```

Сравните с

```
namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("Hello, world!");
        }
    }
}
```

let-определение

```
let x = 1
let x = 2
printfn "%d" x
```

можно читать как

```
let x = 1 in let x = 2 in printfn "%d" x
```

и понимать как подстановку λ-терма

let-определение, функции

```
let powerOfFour x =
    let xSquared = x * x
    xSquared * xSquared
```

- Позиционный синтаксис
  - Отступы строго пробелами
  - Не надо ";"
- Нет особых синтаксических различий между переменной и функцией
- Не надо писать типы
- Не надо писать *return*

Вложенные let-определения

```
let powerOfFourPlusTwoTimesSix n =
    let n3 =
        let n1 = n * n
        let n2 = n1 * n1
        n2 + 2
    let n4 = n3 * 6
    n4
```

- *n3* — не функция!
- Компилятор отличает значения и функции по наличию аргументов
- Значение вычисляется, когда до *let* «доходит управление», функция — когда её вызовут. Хотя, конечно, функция — тоже значение.

Типы

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

F# Interactive

```
val f : x:int -> int
```

Каждое значение имеет тип, известный во время компиляции

Элементарные типы

- *int*
- *double*
- *bool*
- *string*
- ... (.NET)
- *unit* — тип из одного значения, (). Аналог void.

Кортежи (tuples)

```
let site1 = ("scholar.google.com", 10)
let site2 = ("citeseerx.ist.psu.edu", 5)
let site3 = ("scopus.com", 4)
let sites = (site1, site2, site3)

let url, relevance = site1
let site1, site2, site3 = sites
```

Value Tuples

```
let origin = struct (0, 0)

let displace struct (x, y) struct (dx, dy)
    = struct (x + dx, y + dy)

displace origin struct (1, 1)
```

Лямбды

```
let primes = [2; 3; 5; 7]
let primeCubes = List.map (fun n -> n * n * n) primes
```

F# Interactive

```
> primeCubes;;
val it : int list = [8; 27; 125; 343]
```

```
let f = fun x -> x * x
let n = f 4
```

Списки

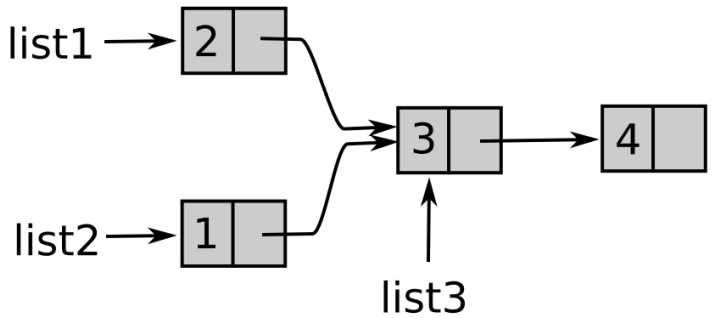
Синтаксис	Описание	Пример
$[]$	Пустой список	$[]$
$[expr; \ldots; expr]$	Список с элементами	$[1; 2; 3]$
$expr :: list$	cons, добавление в голову	$1 :: [2; 3]$
$[expr .. expr]$	Промежуток целых чисел	$[1..10]$
$[for\ x\ in\ list \rightarrow expr]$	Генерированный список	$[for\ x\ in\ 1..99 \rightarrow x * x]$
$list @ list$	Конкатенация	$[1; 2] @ [3; 4]$

Примеры работы со списками

```
let oddPrimes = [3; 5; 7; 11]
let morePrimes = [13; 17]
let primes = 2 :: (oddPrimes @ morePrimes)

let printFirst primes =
    match primes with
    | h :: t -> printfn "First prime in the list is %d" h
    | [] -> printfn "No primes found in the list"
```

Устройство списков




```
let list3 = [3; 4]
let list1 = 2 :: list3
let list2 = 1 :: list3
```

- Списки немутабельны
- Cons-ячейки, указывающие друг на друга
- cons за константное время, @ — за линейное

Операции над списками

Модуль Microsoft.FSharp.Collections.List

Функция	Описание	Пример	Результат
List.length	Длина списка	*List.length* $[1;2;3]$	3
List.nth	n-ый элемент списка	*List.nth* $[1;2;3]$ 1	2
List.init	Генерирует список	*List.init* 3($fun\ i \rightarrow i * i$)	$[0;1;4]$
List.head	Голова списка	*List.head* $[1;2;3]$	1
List.tail	Хвост списка	*List.tail* $[1;2;3]$	$[2;3]$
List.map	Применяет функцию ко всем элементам	*List.map* ($fun\ i \rightarrow i * i$) $[1;2;3]$	$[1;4;9]$
List.filter	Отбирает нужные элементы	*List.filter* ($fun\ x \rightarrow x$ % $2 <> 0$) $[1;2;3]$	$[1;3]$
List.fold	”Свёртка”	*List.fold* ($fun\ x\ acc \rightarrow acc * x$) 1 $[1;2;3]$	6
List.zip	Делает из двух списков список пар	*List.zip* $[1;2]$ $[3;4]$	$[(1,3);(2,4)]$

Тип Option

Либо *Some что-то*, либо *None*, представляет возможное отсутствие значения.

```
let people = [ ("Adam", None); ("Eve" , None);
   ("Cain", Some("Adam","Eve"));
   ("Abel", Some("Adam","Eve")) ]

let showParents (name, parents) =
   match parents with
   | Some(dad, mum) ->
      printfn "%s, father %s, mother %s" name dad mum
   | None -> printfn "%s has no parents!" name
```

Рекурсия

```
let rec length l =
    match l with
    | [] -> 0
    | h :: t -> 1 + length t

let rec even n = (n = 0u) || odd(n - 1u)
and odd n = (n <> 0u) && even(n - 1u)
```

Каррирование, частичное применение

```
let shift (dx, dy) (px, py) = (px + dx, py + dy)
let shiftRight = shift (1, 0)
let shiftUp = shift (0, 1)
let shiftLeft = shift (-1, 0)
let shiftDown = shift (0, -1)
```

F# Interactive

```
> shiftDown (1, 1);;
val it : int * int = (1, 0)
```

Зачем — функции высших порядков

```
let lists = [[1; 2]; [1]; [1; 2; 3]; [1; 2]; [1]]
let lengths = List.map List.length lists
```

или

```
let lists = [[1; 2]; [1]; [1; 2; 3]; [1; 2]; [1]]
let squares = List.map (List.map (fun x -> x * x)) lists
```

Функции стандартной библиотеки стараются принимать список последним, для каррирования

Оператор |>

Pipe forward

```
let (|>) x f = f x

let sumFirst3 ls = ls |> Seq.take 3 |> Seq.fold (+) 0
```

вместо

```
let sumFirst3 ls= Seq.fold (+) 0 (Seq.take 3 ls)
```

Оператор >>

Композиция

```
let (>>) f g x = g (f x)

let sumFirst3 = Seq.take 3 >> Seq.fold (+) 0
let result = sumFirst3 [1; 2; 3; 4; 5]
```

Операторы <| и <<

Pipe-backward и обратная композиция

```
let (<|) f x = f x
let (<<) f g x = f (g x)
```

Зачем? Чтобы не ставить скобки:

```
printfn "Result = %d" <| factorial 5
```

Использование библиотек .NET

```
open System.Windows.Forms

let form = new Form(Visible = false, TopMost = true, Text = "Welcome to F#")
let textB = new RichTextBox(Dock = DockStyle.Fill, Text = "Some text")
form.Controls.Add(textB)

open System.IO
open System.Net

/// Get the contents of the URL via a web request
let http(url: string) =
    let req = System.Net.WebRequest.Create(url)
    let resp = req.GetResponse()
    let stream = resp.GetResponseStream()
    let reader = new StreamReader(stream)
    let html = reader.ReadToEnd()
    resp.Close()
    html

textB.Text <- http("http://www.google.com")

form.ShowDialog () |> ignore
```

Сопоставление шаблонов

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

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

F# — не Prolog

Не получится писать так:

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

Нужно так:

```
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”	$[x] \mid [\text{“}X\text{“} ; x]$
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*

Юнит-тестирование в F#

- Работают все дотнетовские библиотеки (NUnit, MsTest и т.д.)
- Есть обёртки, делающие код тестов более “функциональным” (FsUnit)
- Есть чисто F#-овские штуки: FsCheck, Unquote
  - на самом деле, не совсем F#-овские, но в C# такого нет

FsUnit, пример

```
module ``Project Euler - Problem 1`` =
    open NUnit.Framework
    open FsUnit

    let GetSumOfMultiplesOf3And5 max =
        seq{3 .. max - 1}
        |> Seq.fold(fun acc number ->
                (if (number % 3 = 0 || number % 5 = 0) then
                    acc + number else acc)) 0

    [<Test>]
    let ``Sum of multiples of 3 and 5 to 10 should return 23`` () =
        GetSumOfMultiplesOf3And5(10) |> should equal 23
```

FsUnit, матчеры

```
1 |> should equal 1
1 |> should not' (equal 2)
10.1 |> should (equalWithin 0.1) 10.11
"ships" |> should startWith "sh"
"ships" |> should not' (endWith "ss")
"ships" |> should haveSubstring "hip"
[1] |> should contain 1
[] |> should not' (contain 1)
anArray |> should haveLength 4

(fun () -> failwith "BOOM!" |> ignore)
    |> should throw typeof<System.Exception>

shouldFail (fun () -> 5/0 |> ignore)
```

FsUnit, ещё матчеры

```
true |> should be True
false |> should not' (be True)
"" |> should be EmptyString
null |> should be Null

anObj |> should not' (be sameAs otherObj)

11 |> should be (greaterThan 10)
10.0 |> should be (lessThanOrEqualTo 10.1)

0.0 |> should be ofExactType<float>
1 |> should not' (be ofExactType<obj>)
```

FsUnit, и ещё матчеры

```
Choice<int, string>.Choice1Of2(42) |> should be (choice 1)

"test" |> should be instanceOfType<string>
"test" |> should not' (be instanceOfType<int>)

2.0 |> should not' (be NaN)

[1; 2; 3] |> should be unique

[1; 2; 3] |> should be ascending
[1; 3; 2] |> should not' (be ascending)
[3; 2; 1] |> should be descending
[3; 1; 2] |> should not' (be descending)
```

FsCheck

Библиотека, которая берёт функцию и закидывает её случайно сгенерёнными тестами:

```
open FsCheck

let revRevIsOrig (xs:list<int>) = List.rev(List.rev xs) = xs

Check.Quick revRevIsOrig
// Ok, passed 100 tests.

let revIsOrig (xs:list<int>) = List.rev xs = xs
Check.Quick revIsOrig
// Falsifiable, after 2 tests (2 shrinks) (StdGen (338235241,296278002)):
// Original:
// [3; 0]
// Shrunk:
// [1; 0]
```

Unquote

Вообще интерпретатор F#-а, очень полезный для тестирования:

```
[<Test>]
let ``Unquote demo`` () =
    test <@ ([3; 2; 1; 0] |> List.map ((+) 1)) = [1 + 3..1 + 0] @>

// ([3; 2; 1; 0] |> List.map ((+) 1)) = [1 + 3..1 + 0]
// [4; 3; 2; 1] = [4..1]
// [4; 3; 2; 1] = []
// false
```

Foq

Ну и, конечно же, mock-объекты:

```
[<Test>]
let ``Foq demo`` () =
    let mock = Mock<System.Collections.Generic.IList<int>>()
                .Setup(fun x -> <@ x.Contains(any()) @>).Returns(true)
                .Create()

    mock.Contains 1 |> Assert.True
```