Язык программирования F#

Доклад подготовил Павлов А.Е.

Краткий экскурс

- F# - это мультипарадигмальный язык программирования из семейства языков .Net Framework.
- Поддерживаемые парадигмы: функциональная, императивная и объектноориентированная.
- Во многом F# похож на OCaml с той лишь разницей, что реализован поверх библиотек и среды выполнения .NET.
- F# включен в Microsoft Visual Studio начиная с верси 2010.
- Существуют так же компиляторы для Mac и Linux.

Немного истории

- В 2002 году команда разработчиков по руководством Дона Саймона приступила к реализации функциональной парадигмы на платформе .NET.
- В 2005 году появилась первая версия нового языка программироваия, поддерживающего функциональное программирование – F#. С тех пор вокруг F# стало формироваться сообщество. За счёт поддержки функциональной парадигмы язык оказался востребован в научной сфере и финансовых организациях. Во многом благодаря этому Microsoft решила перевести F# из статуса исследовательских проектов в статус поддерживаемых продуктов и поставить его в один ряд с основными языками платформы .NET. 12 апреля 2010 года свет увидела новая версия флагманского продукта для разработчиков — Microsoft Visual Studio 2010, которая поддерживает разработку на F# прямо из коробки.

Отличительные особенности

- F# относится к языкам со статической типизацией, т.е. тип данных определяется на этапе компиляции и не может быть изменен.
- В F# действует строгая типизация, т.е. неявное приведение типов полностью отсутствует (это помогает избежать ошибок, связанных с приведением типов).
- В F# все значения по умолчанию являются константами, т.е. проинициализированное значение невозможно больше изменить. Если нужно сделать переменную, это нужно декларативно объявить.
- В F# функции – такие же полноправные типы данных.
- В F# пробелы имеют особое значение. Каждая новая инструкция должна начинаться с новой строки, а для выделения отдельного блока кода используются пробелы (вместо {} для C - подобных языков).

Элементарные типы данных
Числовые типы

Тип	Тип .NET	Диапазон
byte	System.Byte	0 – 255
sbyte	System.SByte	-128 – 127
int16	System.Int16	-32768 – 32767
uint16	System.UInt16	0 – 65535
Int, int32	System.Int32	-2^31 – 2^31 - 1
uint32	System.UInt32	0 – 2^32 - 1
Int64	System.Int64	-2^63 – 2^63 - 1
uint64	System.UInt64	0 – 2^64 - 1
float	System.Double	Вещественный тип двойной точности
float32	System.Single	Вещественный тип одинарной точности
decimal	System.Decimal	Вещественный тип фиксированной точности

Элементарные типы данных
Тип Bigint

- Если приходится иметь дело с большими целыми числами (больше 2^64), то можно воспользоваться типом *Bigint*.
- Данный тип позволяет использовать целые значения произвольной длины.
- Литералы типа *Bigint* должны оканчиваться символом I, например: *1024I, 2233455566778890998761*.
- Несмотря на то, что операции с типом *Bigint* значительно оптимизированы, тем не менее они выполняются существенно медленнее, чем операции над значениями элементарных числовых типов.

Элементарные типы данных Символы и строки

- Платформа .NET основана на использовании Юникода. Поэтому символы и строки в F# кодируются знаками Юникода.
- Отдельные символы обрамляются одинарными кавычками: *'f'*, *'в'*.
- Строки обрамляются двойными кавычками: *"Это строка F#"*.
- Спец. символы: спецсимволы начинаются с символа '\', например: '\n', '\b', '\r'.
- Длинную строку можно разбить на несколько. Символ разбиения – '\':

*"Это очень длинная *

*строка, которая, как оказалось, *

не поместилась на одной строке"

Элементарные типы данных Логический тип

- Логическим типом является тип *bool* (*System.Boolean*), принимающий всего два значения: *true* и *false*.
- Булевы операторы: && - логическое и, || - логическое или, not – логическое отрицание.
- Операторы сравнения, возвращающие логический тип данных: >, <, >=, <=, =, <>. Обратите внимание, что сравнение на равенство определено символом «=», а не символом «==», как в языках C, C++, C#.

Компилятор и командный интерпретатор

- F# - «компилируемый» язык программирования. То есть код F# компилируется в так называемый CIL – код (Common Intermediate Language – байт код, в который компилируются все языки платформы .NET).
- Но наряду с компилятором есть так же интерактивный интерпретатор, позволяющий налету компилировать и запускать на исполнение отдельные инструкции на языке F#. Интерпретатор позволяет проверять и тестировать отдельные участки кода при разработке больших проектов либо использовать скрипты для расширения функциональных возможностей ОС.
- Если в основном коде команды пишутся с новой строки, то в интерпретаторе помимо этого, чтобы запустить участок кода, нужно последнюю команду закончить двойной точкой с запятой - ;;. Сразу после этого команды, находящиеся до этих двух символов, будут немедленно выполнены.

Объявление значений Оператор связывания

- При объявлении значений либо функций обычно не используется имя типа. Компилятор вычисляет его сам исходя из контекста. Например при присваивании значению “x” величины 2, компилятор укажет его тип как *int* (для любых целых чисел тип *int* является типом по умолчанию). При объявлении функций компилятор определяет, какой тип данных возвращает функция в зависимости от того, какие типы используются внутри функции и в зависимости от параметров функции.
- Самый часто используемый оператор в языке F# - оператор связывания, позволяющий связать имя значения или функции с его присваиваемым значением:

```
let x = 2          // Объявление значения 2 с именем x, при этом
                   // подразумевается тип int

let f x = x*x      // Объявление функции, возвращающей квадрат аргумента, при
                   // этом параметр функции является типом int, возвращаемое
                   // значение тоже int.
```

Функции в F#

- В отличие от большинства языков программирования, в F#, при объявлении функции, отсутствует оператор возврата значения (return).
- Пример объявления функции:

```
let f x = x * x
```

Тут объявлена функция, принимающая один параметр типа *int* и возвращающая значение квадрата параметра, тоже типа *int*.

Если необходимо, чтобы функция работала со значениями типа *float*, то можно поступить одним из двух способов:

1. `let f x = x * x * 1.0`
2. `let f(x : float) = x * x`

Кортежи

- Кортеж (tuple) – упорядоченная коллекция данных и наиболее простой способ группировки данных в одну структуру.
- Чтобы создать кортеж, достаточно указать список значений через запятую и (необязательно) заключить их в круглые скобки.
- Тип кортежа обозначается с помощью типов его элементов.
- Кортежи могут содержать любое количество элементов любых типов.
- Пример кортежей:

```
let vector = (1, 2, 3)
let zeros = (0, 0.0, '0')
```

Кортежи

- Для извлечения значений из кортежа, содержащего два элемента, можно использовать функции *fst* и *snd*. Функция *fst* возвращает первый элемент кортежа, *snd* – второй.
- Либо можно использовать оператор связывания *let*, за которым должен идти список идентификаторов, разделенных запятой. При этом количество идентификаторов должно быть равно количеству элементов кортежа.
- Примеры:

```
let names = (“Иван”, “Петр”)
let n1 = fst names          // примет значение “Иван”
let n2 = snd names          // примет значение “Петр”
let  vector = (1, 2.5, “str”)
let a1, a2, a3 = vector     // a1 = 1; a2 = 2.5; a3 = “str”
```

Списки

- Кортежи группируют элементы в единую сущность, списки в цепочку элементов.
- Это позволяет обрабатывать элементы списков с помощью агрегатных операторов.
- В отличие от списков в других языках программирования, списки в языке F# имеют весьма ограниченные возможности доступа к элементам и управления ими. Фактически списки поддерживают всего две операции: операция добавления и операция объединения.

Списки

Объявление списков:

```
let lst1 = [1; 2; 3]
let  lst2 = []          // Пустой список
```

Добавление в начало списка:

```
let lst3 = 0::lst       // фактически создается новый список, первым
                        // элементом которого будет 0, затем пойдут остальные
                        // элементы списка lst.
```

Объединение списков:

```
let lst4 = [4; 5; 6]
let lst5 = lst1 @ lst4      //список будет иметь вид: [1; 2; 3; 4; 5; 6]
```

Списки

- Объявление элементов списков в виде перечней значений, разделенных точками с запятой, быстро превращается в утомительное занятие, особенно при работе с длинными списками. Объявить список упорядоченных числовых значений можно с помощью синтаксиса диапазонов.
- Если указано необязательное значение шага, то в результате будет создан список значений в диапазоне между двумя числами с заданным шагом.

```
let lst = [1..10]          // lst = [1; 2; 3; 4; 5; 6; 7; 8; 9; 10]
let lst1 = [10..10..50]    // lst1 = [10; 20; 30; 40; 50]
```

Списки

- Наиболее выразительный способ создания списков заключается в использовании генераторов списков.

```
let x =
    [ let negate x = -x
        for i in 1 .. 10 do
            if i % 2 = 0 then
                yield negate i
            else
        yield i ]
// x = [1; -2; 3; -4; 5; -6; 7; -8; 9; -10]
```

Агрегатные операторы

- Модуль *List* из стандартной библиотеки F# содержит множество методов, упрощающих обработку списков.

Функция *List.map* – это операция проекции, которая создает новый список на основе заданной функции.

Функция *List.reduce* проходит по всем элементам списка и накапливает значение аккумулятора, представляющее собой результат обработки списка к текущему моменту. В случаях, когда требуется использовать произвольный тип аккумулятора, можно воспользоваться функцией *List.fold*.

List.iter, проходит по всем элементам списка и вызывает функцию, переданную ей в виде параметра.

Агрегатные операторы Примеры

```
let squares x = x * x
List.map squares [1 .. 10]          //[1; 4; 9; 16; 25; 36; 49; 64; 81; 100]

let lst = [1..10]                   // lst = [1; 2; 3; 4; 5; 6; 7; 8; 9; 10]
let sum acc item = acc + item
List.reduce sum lst                 // 55

let printing x = printfn "Printing %d" x
List.iter printing [1..3]           // Вывод:
                                    // Printing 1
                                    // Printing 2
                                    // Printing 3
```

Строение программы на языке F#

- В языке F# для приложений, состоящих из одного файла, код выполняется в направлении от начала и до конца (при этом не требуется объявлять специальный главный метод).
- Однако для проектов, состоящих из нескольких файлов, программный код должен делиться на организационные единицы, называемые модулями и пространствами имен.
- По умолчанию компилятор F# помещает весь программный код в *анонимный модуль* с тем же именем, что и имя файла, с заглавным первым символом имени. То есть если в коде имеется значение с именем *value1*, а файл называется *file1.fs*, вы можете обратиться к нему через полностью квалифицированное имя: *File1.value1*.
- Если необходимо использовать какую либо библиотеку .NET, можно использовать инструкцию *open* *<Имя сборки>*, например: *open System.Threading*.

Строение программы на языке F#

- Чтобы явно указать главный метод, следует пометить его как точка входа:

```
// Program.fs
open System
[<EntryPoint>]
let main (args : string[]) =
    let numbers = [1 .. 10]
    // Вернуть 0
    0
```

Программирование в функциональном стиле

- В основе функционального программирования лежит представление о коде в терминах математических функций.
- Вам не нужно быть математиком, чтобы программировать на F#, но некоторые идеи функционального программирования пришли именно из этой науки.
- В функциональном программировании программист описывает, что требуется сделать, а не то, как это сделать.

Пример. Суперпозиция двух функций:

```
let f x = x ** 2.0 + x
let g x = x + 1.0
f(g(3.0))                    // На экран будет выведено 20.0
```

Программирование в функциональном стиле

Лямбда – выражения:

```
List.map (fun i -> -i) [1 .. 10]
// На экран будет выведено [–1; –2; –3; –4; –5; –6; –7; –8; –9; –10]
```

Лямбда – выражение в F# начинается с оператора *fun*, затем идет перечисление аргументов, после символ -> и результат действия выражения.

Замыкание:

```
let powGenerator _base =
    let powOfBase _pow =
        _base ** _pow
    powOfBase
let powOfTwo = powGenerator(2.0)
powOfTwo(3.0)              // На экран будет выведено 8.0
```

Программирование в функциональном стиле

Символьные операторы:

Пример вычисления факториала

```
let rec (!) x =
    if x <= 1 then 1
    else x * !(x - 1)
!5                          // На экран будет выведено 120
```

Программирование в функциональном стиле

Композиция функций. Прямой конвейерный оператор.

Прямой конвейерный оператор определен в системе следующим образом:

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

Эта замысловатая запись позволяет использовать нам суперпозицию нескольких функций в более компактном виде:

Вместо написания: *f(g(h(x)))*

Можно написать: `x |> h |> g |> f`

Прямой конвейерный оператор позволяет переупорядочить параметры функции так, что при вызове функции последний ее параметр указывается первым.

Программирование в функциональном стиле

Пример использования прямого конвейерного оператора. Вычисление размера каталога:

```
open System
open System.IO
let sizeOfFolder folder =
    let getFiles folder =
        Directory.GetFiles(folder, "*.*", SearchOption.AllDirectories)
    folder
    |> getFiles
    |> Array.map(fun file -> new FileInfo(file))
    |> Array.map(fun info -> info.Length)
    |> Array.sum
Console.WriteLine("Размер директории D:\\Temp: {0}", sizeOfFolder("D:\\Temp"))
```

Программирование в функциональном стиле

Композиция функций. Прямой оператор композиции.

Прямой оператор композиции >> объединяет две функции, при этом функция слева вызывается первой:

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

При использовании прямого конвейерного оператора необходимо указать переменную, чтобы «запустить» конвейерную обработку. В нашем последнем примере функция принимала параметр *folder*, который передавался первой функции в конвейере.

Однако при использовании оператора композиции функций никакие параметры не требуется. Мы можем просто объединить все эти функции воедино и получить в результате новую функцию, принимающую параметр и возвращающую результат.

Программирование в функциональном стиле

Пример. Использование прямого оператора композиции.

```
open System
open System.IO
let sizeOfFolderComposed =
    let getFiles folder =
        Directory.GetFiles(folder, "*.*", SearchOption.AllDirectories)
    getFiles
    >> Array.map(fun file -> new FileInfo(file))
    >> Array.map(fun info -> info.Length)
    >> Array.sum
Console.WriteLine("Размер директории D:\\Temp: {0}", sizeOfFolderComposed "D:\\Temp")
```

Программирование в функциональном стиле

Наряду с прямыми операторами конвейера и композиции есть конечно же и обратные.

Обратный конвейерный оператор определяется в системе так:

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

Кажется, что он ничего не меняет.. Но с помощью этого оператора можно изменить порядок вычислений.

Обратный оператор композиции определяется в системе так:

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

Использование обратного оператора композиции позволяет читать текст программы именно так, как он будет выполняться, то есть записывать программный код, который будет читаться в точном соответствии с тем, как он работает.

Программирование в функциональном стиле

Пример использования обратного оператора композиции:

```
let square x = x * x
let negate x = -x
// Использование (>>)изменяет знак квадрата числа
(square >> negate) 10     // Будет получено -100
(square << negate) 10     // Будет получено 100
```

Подведение итогов

Все, что было рассказано – лишь базовый функционал языка. Это лишь малая толика того, что можно делать с помощью языка программирования F#.

Язык F# идеально подходит для решения сложных математических задач, распараллеливания алгоритмов, асинхронного программирования.

А смешивание функциональной и объектно ориентированной парадигм вкупе с библиотеками .NET позволяет разрабатывать сложные программные комплексы, такие как WinForms приложения, WPF приложения, приложения, работающие с базами данных, приложения, использующие системные вызовы API функций операционной системы и COM объектов.

Полезная информация, книги

- Смит К. Программирование на F#. – Пер. с англ. – СПб.: Символ Плюс, 2011. – 448 с.: ил.
- Сошников Д.В. Программирование на F#. – М.: ДМК Пресс, 2011. – 192 с.: ил.
- Интерактивный учебник по Visual F# [Электронный ресурс]. https://msdn.microsoft.com/ru-ru/library/dd233157(v=vs.120).aspx
- Visual F# Development Portal [Электронный ресурс]. https://msdn.microsoft.com/en-us/library/ff730280.aspx

Спасибо за внимание.