Продвинутые концепции F#

SRTP / inline / inlineIfLambda

Похабов Иван 23.Б10-мм

22 мая 2025 г.

SRTP: Statically Resolved Type Parameters

- Механизм F# для написания обобщенных функций, где операции над типами-параметрами разрешаются статически (во время компиляции)
- Используется для операций, обычно недоступных для стандартных дженериков (например, арифметика)
- Требуют использования ключевого слова `inline`

SRTP: ключевые моменты

- `inline`: Функции с SRTP **обязательно** должны быть `inline`
- **Ограничения-члены (Member Constraints):** Указывают, какие статические или инстансные члены должен иметь тип-параметр
  - `when ^T : (static member OperatorName : ^T * ^T -> ^T)`
  - `when ^T : (member MethodName : argType -> returnType)`
  - `when ^T : null`
  - `when ^T : comparison`, `equality`
  - `when ^T : (new : unit -> ^T)` (конструктор по умолчанию)
- **«Шляпные типы» (`^T`):** Часто обозначают SRTP, но не всегда строго обязательны (компилятор может вывести)

SRTP: зачем нужно?

- **Обобщенная арифметика:** Функция `add` для `int`, `float`, `decimal` и т.д., без перегрузок или рефлексии
- **Работа с конструкторами обобщенных типов**
- **Важно:** Может привести к увеличению размера кода (code bloat), так как генерируется специализированный код для каждого типа

SRTP: обобщенное сложение

```fsharp
let inline add (x: ^T) (y: ^T) : ^T =
    x + y

let inline addExplicit (x: ^a) (y: ^a) : ^a =
    (^a : (static member op_Addition : ^a * ^a -> ^a) (x, y))

let intSum = add 10 20                          // ^T выводится как int
let floatSum = add 5.5 2.3                      // ^T выводится как float

printfn "Int sum: %d" intSum
printfn "Float sum: %f" floatSum
// let stringIntSum = add "a" 1 // Ошибка компиляции
```

SRTP: создание объекта

```fsharp
let inline createDefault<'T when 'T: (new : unit -> 'T)> () : 'T =
    new 'T()

type MyCustomClass() =
    member _.Value = "Hello from MyCustomClass"

let mc = createDefault<MyCustomClass>()
printfn "%s" mc.Value

let rnd = createDefault<System.Random>()
printfn "Random int: %d" (rnd.Next(100))
```

`inline` функции

- Ключевое слово `inline` — это директива компилятору F# встроить тело функции непосредственно в место ее вызова
- Это обязательное требование – если скомпилировалось, то функция инлайнится

Преимущества:

- Уменьшение накладных расходов на вызов функции (нет стековых фреймов, копирования аргументов и т.д.)
- Необходимость для SRTP (компилятор должен “видеть” типы в месте вызова)
- Оптимизация лямбда-выражений (тело лямбды может быть встроено, устраняя создание объекта-замыкания)

`inline` функции: когда использовать и осторожность

Когда использовать:

- Для небольших, часто вызываемых служебных функций
- Когда используются SRTP
- В критически важных по производительности участках кода
- В функциях высшего порядка, активно работающих с лямбдами

Когда быть осторожным:

- **Большие функции:** Инлайнинг может привести к увеличению размера кода (code bloat)
- **Рекурсивные функции:** Обычно не могут быть полностью инлайнены

inline функции: пример

```fsharp
let inline square x = x * x

let num = 5
let result = square (num + 1) // -> (5+1)*(5+1) -> 6*6 -> 36

printfn "Square of %d+1 is %d" num result

let inline mapAndSum (projection: 'a -> int) (values: 'a list) =
    values
    |> List.map projection
    |> List.sum

let data = [1; 2; 3]
let sumOfSquares = mapAndSum (fun x -> x * x) data
// Лямбда (fun x -> x*x) также будет инлайнена

printfn "Sum of squares for %A: %d" data sumOfSquares
```

Атрибут [<inlineIfLambda>]

- Атрибут [<inlineIfLambda>] — это «мягкая» форма inline
- Подсказывает компилятору инлайнить функцию **только в том случае, если один или несколько ее аргументов являются лямбда-выражениями**
- **Цель:** Оптимизировать функции высшего порядка, когда они вызываются с лямбдами, без инлайнинга в других случаях

Как это работает:

- Если функция вызывается с лямбдой: инлайнинг тела функции и лямбды устраняет создание объекта-замыкания и косвенный вызов
- Если функция вызывается с уже скомпилированной функцией: инлайнинг может быть невыгоден, и [<inlineIfLambda>] помогает его избежать

Атрибут [<inlineIfLambda>]: пример

```fsharp
let inline processValue (value: 'a) ([<inlineIfLambda>] f: 'a -> 'b) =
    printfn "Processing value..."
    f value

let result1 = processValue 10 (fun x -> x + 1)
// Компилятор, скорее всего, встроит processValue и лямбду:
// printfn "Processing value..."
// let temp = 10 + 1 (тело лямбды)
// ...

let increment (x: int) = x + 1
let result2 = processValue 20 increment
// Компилятор, скорее всего, НЕ встроит processValue,
// а сгенерирует обычный вызов:
// printfn "Processing value..."
// increment(20)

printfn "Result 1: %A" result1
printfn "Result 2: %A" result2
```

Заключение

- **SRTP:** Предоставляют механизм для написания дженерик функций
- `inline`: Ключевой инструмент для оптимизации, уменьшения накладных расходов на вызовы функций и основа для SRTP и эффективной работы с лямбдами
- `[<inlineIfLambda>]`: Позволяет проводить целевую оптимизацию функций высшего порядка, инлайня их только при использовании с лямбда-аргументами