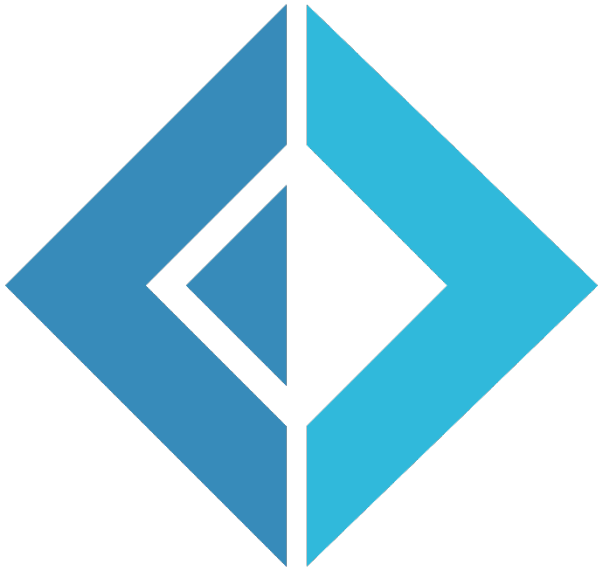
Реактивное функциональное программирование на F#

Дмитрий Сошников {Microsoft | DX | Technical Evangelist}

О чем будет рассказ



F#

Реактивное программирование

Обо мне

> Microsoft
> МФТИ, НИУ ВШЭ, МАИ

> Автор книги: Функциональное программирование на F#

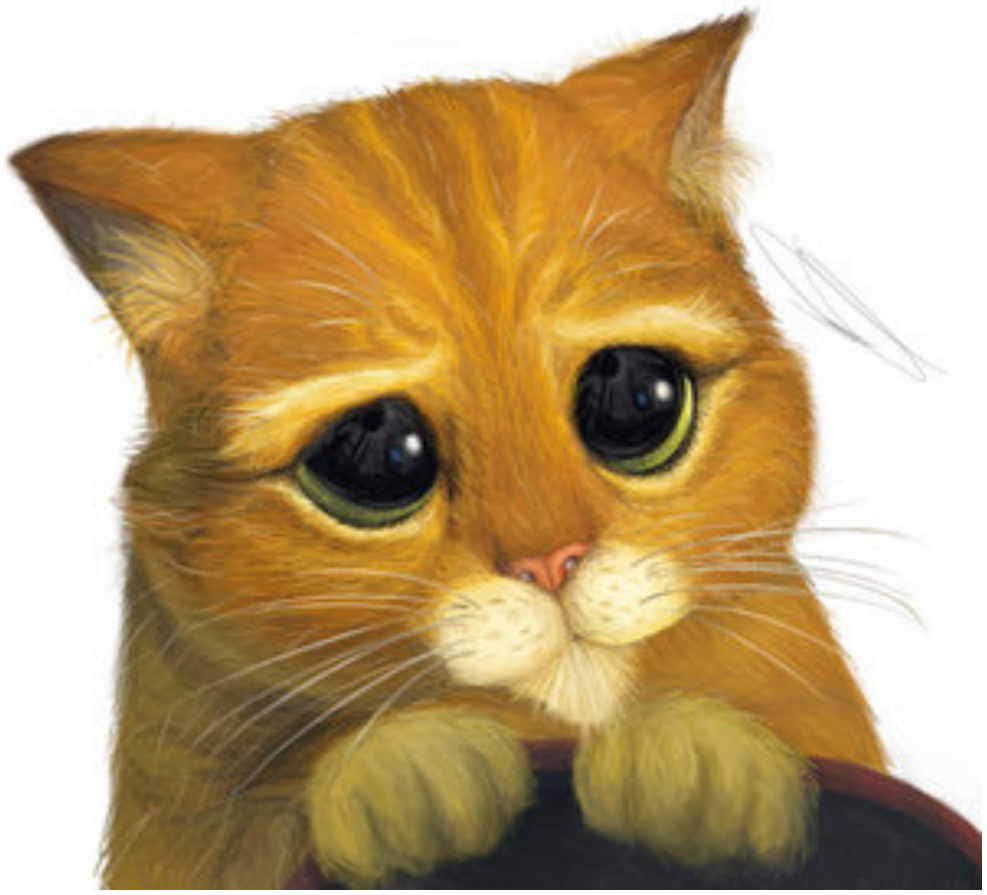
Нелегкий выбор

$$\begin{array}{ccc} TA & \xrightarrow{Tg} & TB \\ {\scriptstyle \eta_{TA}}\downarrow & & \downarrow{\scriptstyle \eta_{TB}} \\ T^2A & \xrightarrow{T^2g} & T^2B \\ {\scriptstyle \mu_A}\downarrow & & \downarrow{\scriptstyle \mu_B} \\ TA & \xrightarrow[Tg]{} & TB \end{array} \qquad 1_{TA} \qquad 1_{TB}$$

В этой лекции вы услышите про монады и корекурсию

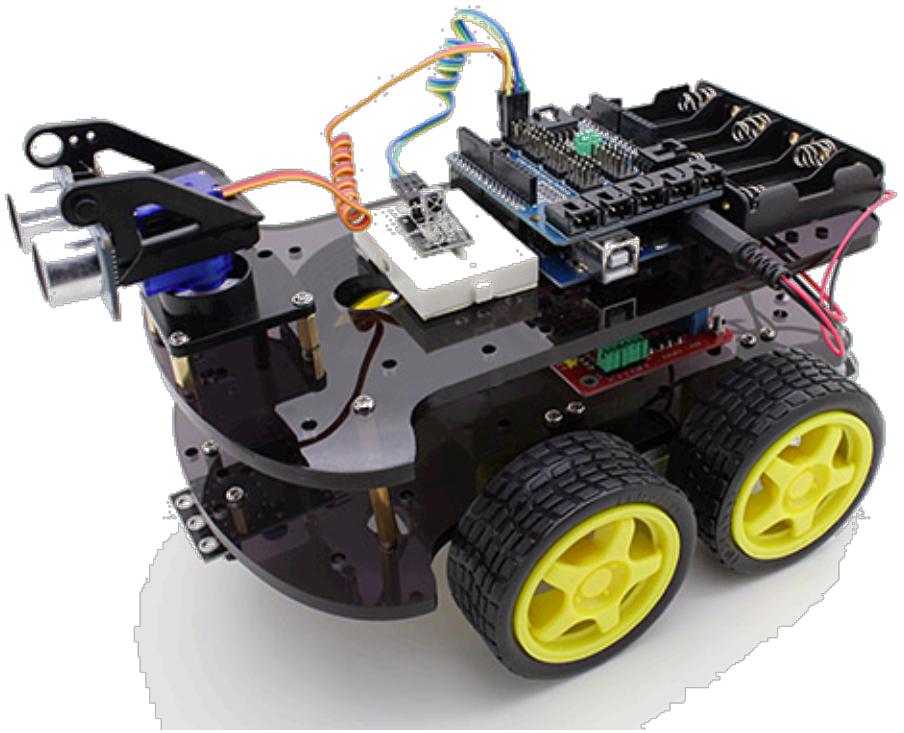
«Используйте F#, или вы сделаете котёнка несчастным…»

Д.Сошников



«Что это за презентация без котиков и роботов…»

Д.Сошников



Введение в F#

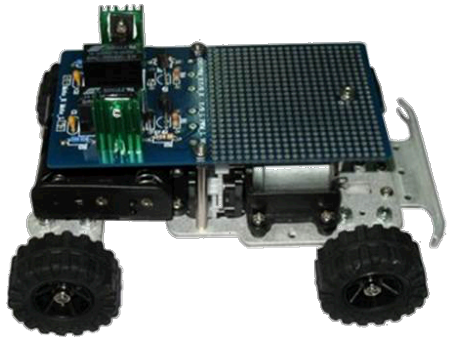
Надеюсь, меньше чем не 80% выступления…

Все ли знают про F#?

since 2010…

F# - Open Source

http://fsharp.org



Microsoft Azure

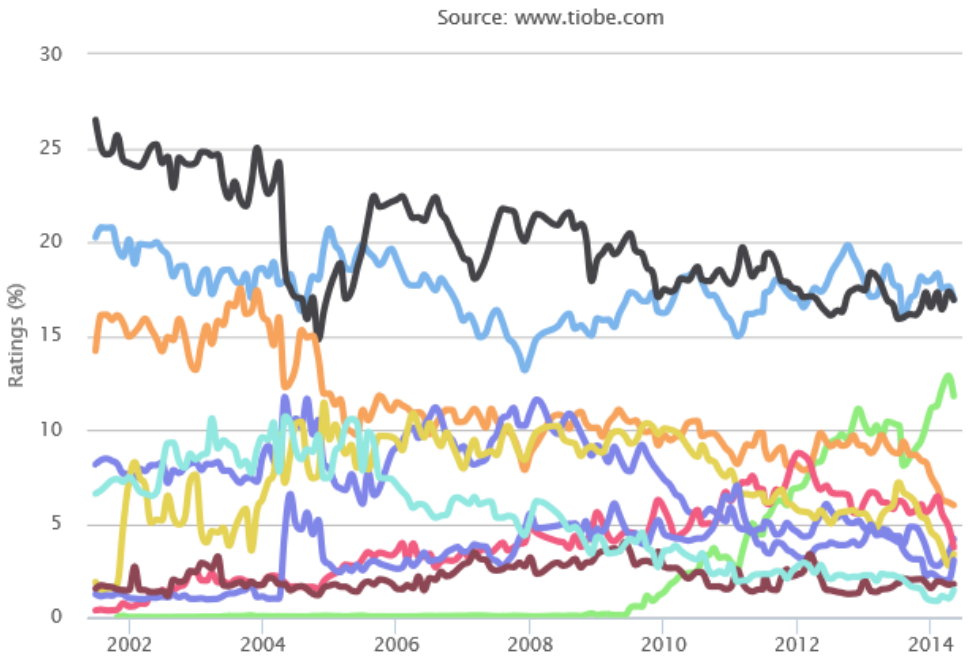
Certified

F# - самый популярный функциональный язык программирования…

TIOBE Programming Language Index

F# - #16
Lisp - #23
ML - #27
Scala - #31
Scheme - #37
Haskell - #38
Erlang - #42
Prolog - #43

F# - самый популярный функциональный язык программирования
F# - самый быстро-растущий по популярности язык
F# - также популярен, как Паскаль/Delphi



Введение и разминка мозга

Вычисление факториала на F#

Очевидно

```
let rec fact n =
  if n=1 then 1 else n*fact(n-1)
```

Симпатично

```
let rec fact = function
  | 1 -> 1
  | n -> n*fact(n-1)
```

Эффективно

```
let fact =
  let rec fact' acc = function
     | 1 -> acc
     | n -> fact' (n*acc) (n-1)
  fact' 1
```

С продолжением

```
let fact =
  let rec fact' f = function
     | 1 -> f 1
     | n -> fact' (f>>((*)n)) (n-1)
  fact' (fun x->x)
```

Во всех случаях – без побочных эффектов

Введение и дальнейшая разминка мозга

Вычисление факториала на F#

Коротко

```
let fact n = [1..n] |> List.reduce (*)
```

Каррировано

```
let fact = (..) 1 >> Seq.reduce (*)
```

Лениво

```
let fact n = Seq.initInfinite (fun x->x+1)
                  |> Seq.scan (*) 1 |> Seq.nth n
```

По т-ме Тарского

```
let rec fix f x = f (fix f) x
let fact = fix (fun f x -> if x=1 then 1 else x*f(x-1))
```

Извращенно-комбинаторно

```
let s f g x = f x (g x)
let k x y   = x
let b f g x = f (g x)
let c f g x = f x g
let rec y f x = f (y f) x
let cond p f g x = if p x then f x else g x
let pred x = x-1
let fact = y (b (cond ((=) 0) (k 1)) (b (s (*)) (c b pred)))
```

Во всех случаях – без побочных эффектов

Конкурс:

До конца лекции придумайте еще один способ вычисления факториала и получите конфетку!

Мощь функционального программирования

- Case Study: комбинаторные парсеры

```
let num = pint32
let list x = (pstring "[" >>. sepBy num (pstring ",") .>> pstring "]") x
CharParsers.run list "[1,2,3,4]“
```

```
let num_ws = pint32 .>> spaces
let str_ws s = pstring s .>> spaces
let list x = (str_ws "[" >>. sepBy num_ws (str_ws ",") .>> str_ws "]") x
CharParsers.run list "[1, 2, 3,4 ]“
```

ЕЯ-интерфейс для робота

```
type Command =
  | Forward of int
  | Back of int
  | Left
  | Right
  | Stop
```

```
let s x = pstring x .>> spaces
let rec sl = function
   | [x] -> s x
   | h::t -> s h >>. sl t
let K x _ = x
let num = pint32 .>> spaces
```

```
let cmdsep = s "then" <|> sl ["after";"that"]
let forward = (s "forward" <|> sl ["go";"forward"] |> attempt) >>. (num |>> Forward)
let back = (s "back" <|> sl ["go";"back"] |> attempt) >>. (num |>> Back)
let left = (s "left" <|> sl ["turn";"left"] |> attempt) |>> (K Left)
let right = (s "right" <|> sl ["turn";"right"]) |>> (K Right)
let stop = (s "stop" <|> sl ["go";"to";"sleep"]) |>> (K Stop)
let simplecommand = (s "please" <|> sl ["i";"beg";"you"] <|> s "")
                    >>. (forward <|> back <|> left <|> right <|> stop)
let command = sepBy simplecommand cmdsep .>> eof

let ParseCommand (s:string) = s.ToLower()
   |> CharParsers.run command
   |> function
       | Success(res,_,_) -> res
       | _ -> []

ParseCommand "please turn right after that forward 10 then i beg you turn left then stop"
```

Разные стили обработки данных

Списки, последовательности, события…

Обработка данных: списки

map

```
[1..10] |> List.map (fun x->x*x)
```

reduce
fold

```
[1..10] |> List.map (fun x->x*x) |> List.reduce (+)

[1..10] |> List.map (fun x->x*x) |> List.fold (+) 0
```

filter

```
[1..10] |> List.filter (fun x-> x%3=0)
```

f

Пример

- Учимся определять позитив

```
let read fn = File.OpenText(fn).ReadToEnd()
let set fn =
     (read fn).Split([|'\n';'\r';' '|])
     |> Array.filter (fun x->x.Length>0)
     |> Array.fold (fun acc x -> Set.add x acc) Set.empty
let positive = set @"d:\books\positive.txt"
let negative = set @"d:\books\negative.txt"
```

```
let wt w =
   if Set.contains w positive then 1
   else if Set.contains w negative then -1
   else 0

let positivity (s:string) =
   s.GoodSplit()
   |> Array.fold (fun acc s -> acc+wt s) 0
```

swap(wt>>(+))

Обработка данных: последовательности

map

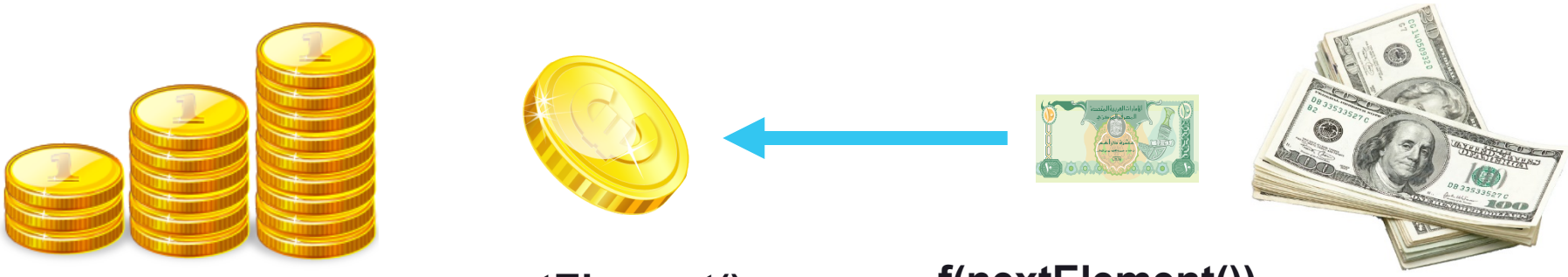
```
{1..10} |> Seq.map (fun x->x*x)
```

reduce
fold

```
{1..10} |> Seq.map (fun x->x*x) |> Seq.reduce (+)

{1..10} |> Seq.map (fun x->x*x) |> Seq.fold (+) 0
```

filter

```
{1..10} |> Seq.filter (fun x-> x%3=0)
```



nextElement()

f(nextElement())

Пример: корекурсия

- Какое минимальное число Фибоначчи делится на 256?

```
Seq.unfold (fun (u,v) -> Some(u,(u+v,u))) (1I,1I)
  |> Seq.filter(fun x->x%256I=0I) |> Seq.head
```

```
let rec fibseq = seq {
                       yield (1I,1I)
                       yield! Seq.map (fun (u,v) -> (u+v,u)) fibseq
                     }

fibseq |> Seq.map fst |> Seq.filter(fun x->x%256I=0I) |> Seq.head
```

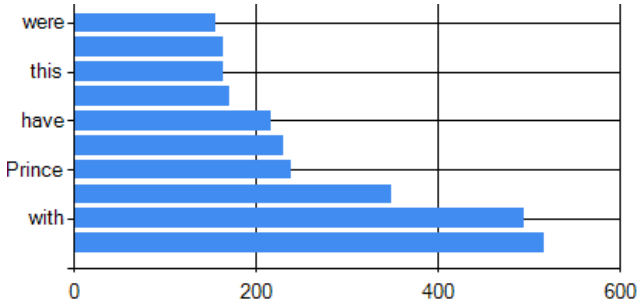
Здесь мы имеем дело с бесконечной последовательностью чисел Фибоначчи! Математики могут быть счастливы!

Программисты же увидят новый для себя вид рекурсии – корекурсия!

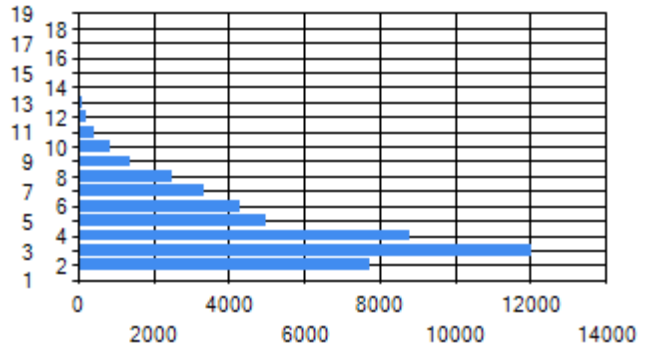
Исследуем творчество Л.Н.Толстого

```
let ReadLines fn =
  seq { use inp = File.OpenText fn in
        while not(inp.EndOfStream) do
            yield (inp.ReadLine())
      }
let file = ReadLines @"d:\books\wap_1.txt"
```

```
file |> Seq.collect (fun s->s.GoodSplit()) |> Seq.filter (fun s-> s.Length>3)
     |> Seq.groupBy (fun x->x)  |> Seq.map (fun (u,v) -> (u, Seq.length v))
     |> Seq.sortBy (fun (u,v) -> -v) |> Seq.take 10 |> FSharpChart.Bar
```



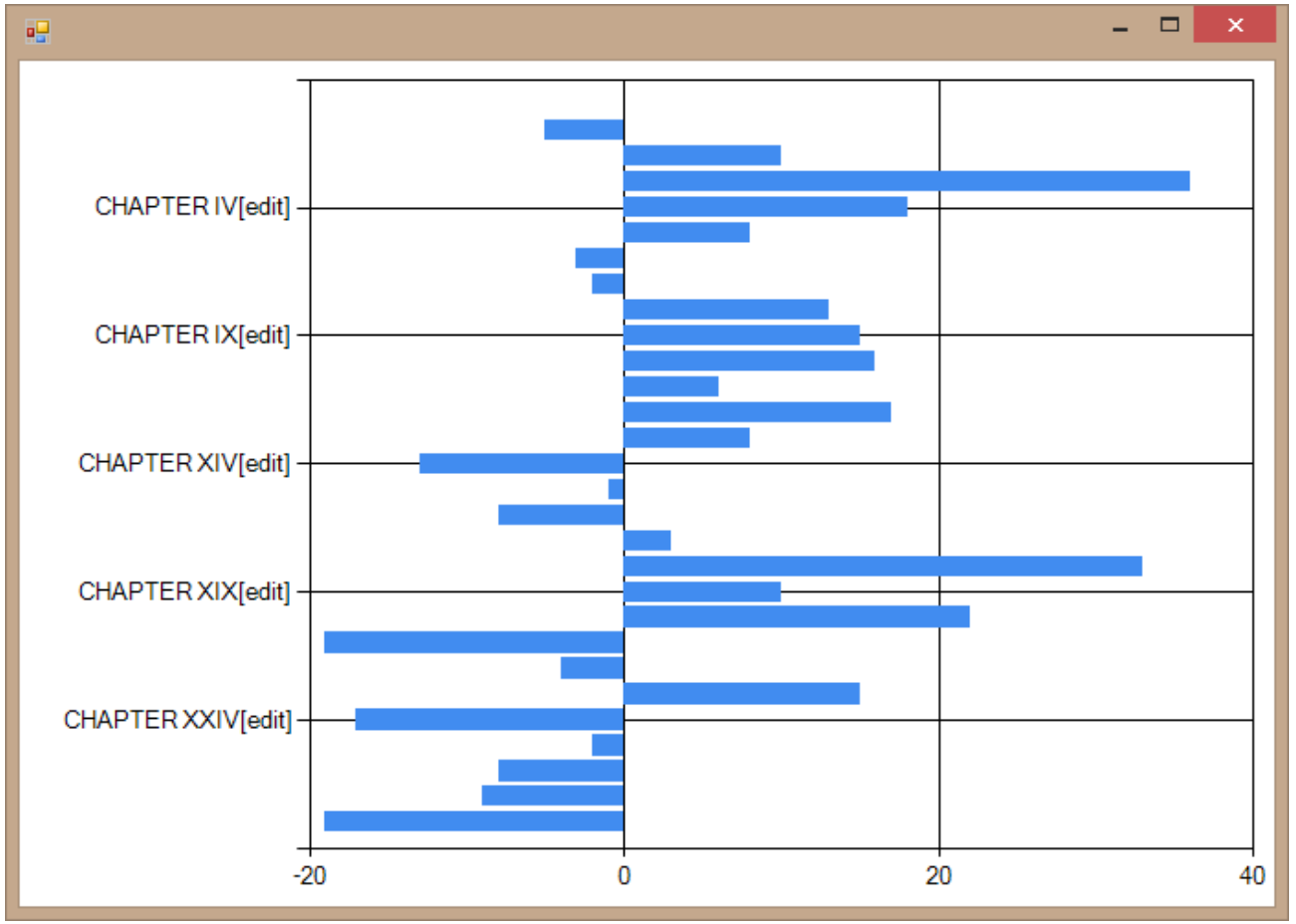

```
file |> Seq.collect (fun s->s.GoodSplit()) |> Seq.filter (fun s-> s.Length>1)
     |> Seq.groupBy (fun x->x.Length)
     |> Seq.map (fun (u,v) -> (u, Seq.length v))
     |> FSharpChart.Bar
```



Вывод:

Война и Мир – добрый роман!

(по крайней мере в переводе на английский)



Бонус: Type Providers

- Type Provider – это:
  - design-time-компонент, предоставляющий вычисляемое пространство типов/классов и методов
  - расширение компилятора / IDE
  - адаптер между источниками данных и языками .NET
- Обеспечивает:
  - Построение типов схемы «на лету» для статической типизации
  - Автоматическое обновление схемы на этапе сборки

Главный вопрос:

А если мы хотим обрабатывать не готовые данные, а возникающие по ходу работы программы?

- Доброта сообщений в твиттере
- Асинхронные данные, приходящие из интернет
- Данные с датчиков робота

Еще один стиль обработки данных: реактивное функциональное программирование

```
let x = new TwitterStreamSample(…)

x.NewFeed
   |> Event.map (fun x -> JsonConvert.DeserializeObject<tweet>(x))
   |> Event.filter (fun x -> x.lang="en")
   |> Event.map (fun x-> positivity x.text)
   |> Event.scan (fun acc x -> acc+x)
   |> FSharpChart.FastLine

x.StartListening()
```

Event<t>

Экскурс: параллельное программирование

```
let task1 = async { return 10+10 }
let task2 = async { return 20+20 }
Async.Run (Async.Parallel [ task1; task2 ])
```

```
let map' func items =
    let tasks =
        seq {
            for i in items -> async { return (func i) } }
    Async.RunSynchronously (Async.Parallel tasks)
```

```
List.map (fun x -> fib(x)) [1..30]
map' (fun x -> fib(x)) [1..30]
```

Экскурс: асинхронное программирование

```
let httpAsync (url:string) =
        async {
            let req = WebRequest.Create(url)
            let! resp = req.AsyncGetResponse()
            use stream = resp.GetResponseStream()
            use reader = new StreamReader(stream)
            let! text = Async.AwaitTask(reader.ReadToEndAsync())
            return text }
```


```
["http://www.yandex.ru";"http://www.bing.com";"http://www.google.com"]
 |> List.map httpAsync
 |> Async.Parallel
 |> Async.RunSynchronously
 |> Array.map (fun x->x.Length)
```

```
["http://www.yandex.ru";"http://www.bing.com";"http://www.google.com"]
 |> List.map (fun x -> async { let! s = httpAsync x
                               return s.Length })
 |> Async.Parallel
 |> Async.RunSynchronously
```

Анатомия Event

- let evt = new Event<T>()
- evt.Publish – интерфейс для подписки на события
- evt.Publish |> Event.map f |> Event.filter p

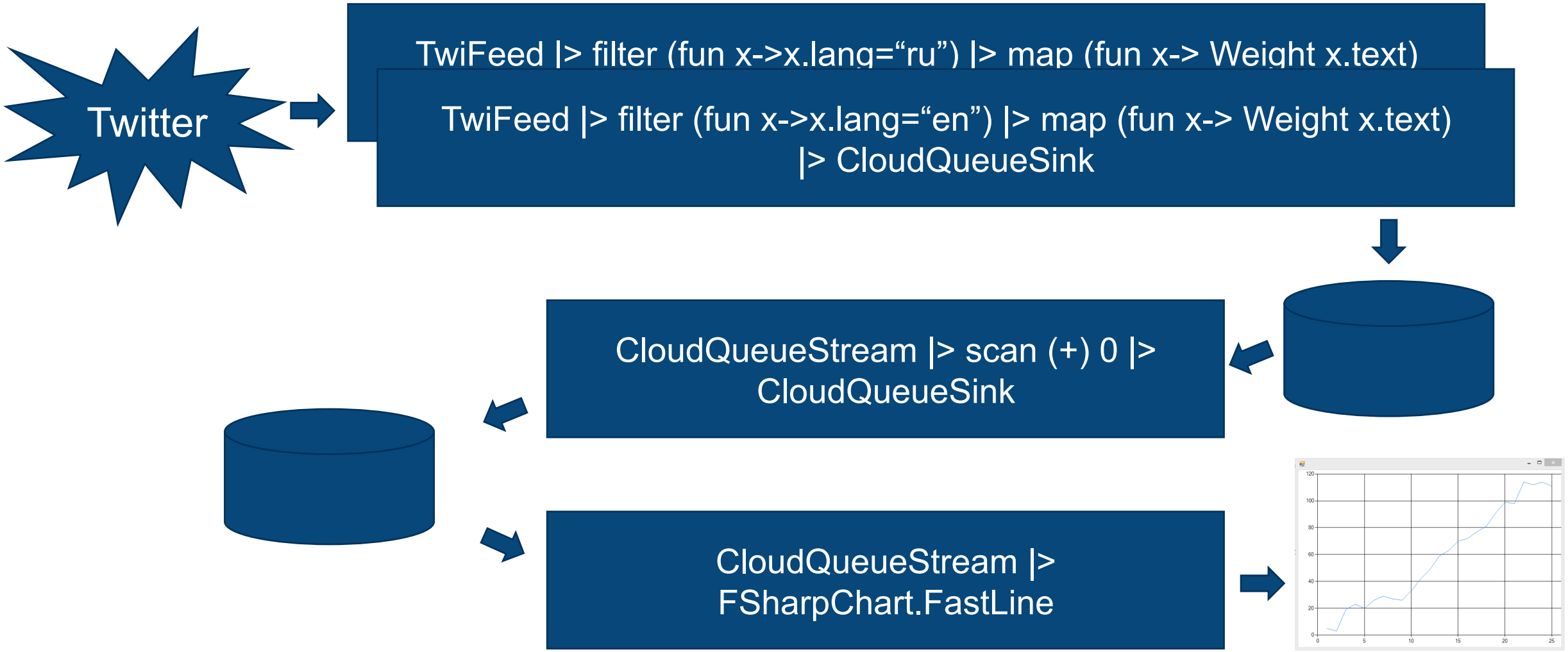
  |> Event.add(fun x -> …)
- evt.Trigger(x : T) – инициирует событие

Интереснее: настроение твиттера

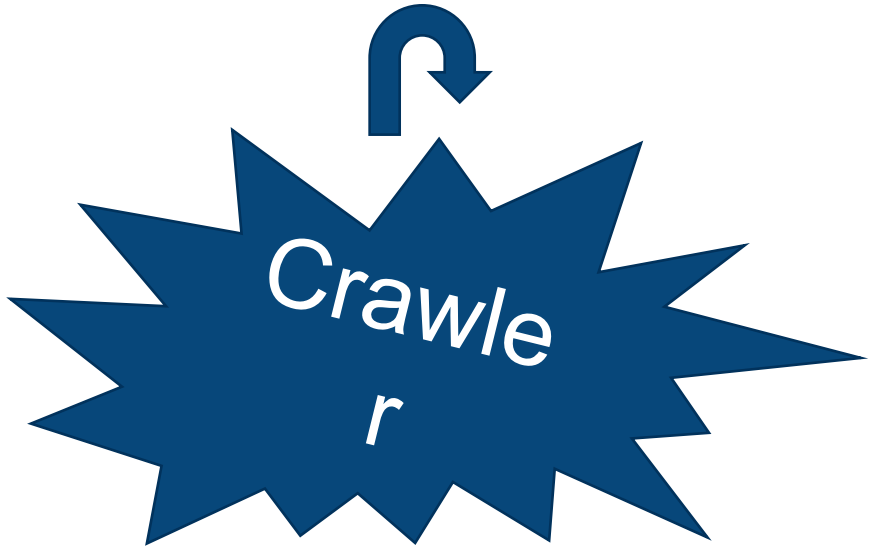
```
type TwitterStreamSample(token,secret) =
    let event = new Event<_>()
    let listener =
        async {
                let req = WebRequest.Create(streamSampleUrl) // more OAUTH magic
                use! resp = req.AsyncGetResponse()
                use stream = resp.GetResponseStream()
                use reader = new StreamReader(stream)
                while ((not reader.EndOfStream)&&(not IsCancellationRequested)) do
                    let z = reader.ReadLine()
                    event.Trigger text
                }
    member this.StartListening() = Async.Start(listener)
    member this.StopListening() = Async.CancelDefaultToken()
    member this.NewFeed = event.Publish
```

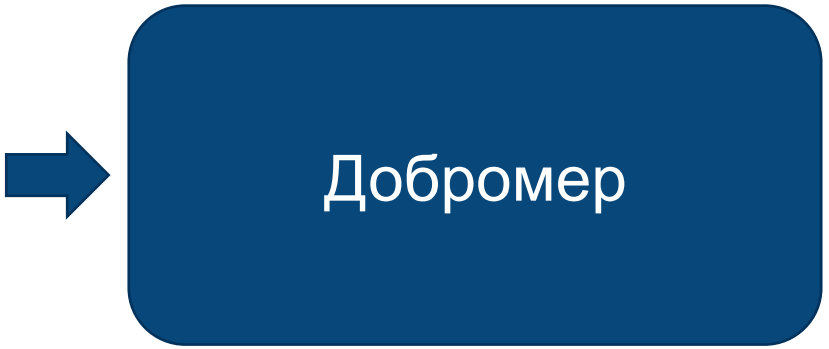
```
let x = new TwitterStreamSample(…)

x.NewFeed
   |> Event.map (fun x -> JsonConvert.DeserializeObject<tweet>(x))
   |> Event.filter (fun x -> x.lang="en")
   |> Event.map (fun x-> positivity x.text)
   |> Event.scan (fun acc x -> acc+x)
   |> FSharpChart.FastLine
```

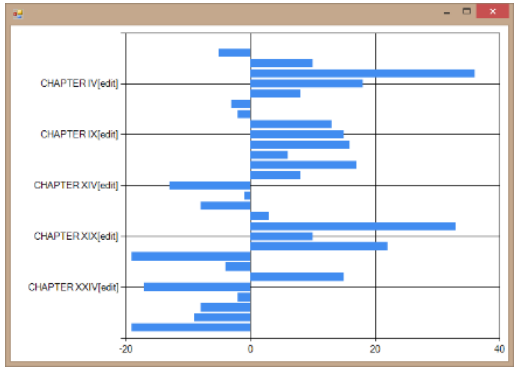
Распределенное реактивное программирование в облаке



Индексируем интернет








```
let cr = new Crawler()

cr.NewFeed
 |> Event.map fst
 |> Event.map positivity
 |> Event.scan (+) 0
 |> FSharpChart.FastLine

cr.StartIndexing("http://yandex.ru")

cr.StopIndexing()
```

Индексируем интернет

```
let Crawl url =
        async {
            let! s = httpAsync url
            return (s,s.Split("") |> Seq.filter (fun s-> s.StartsWith("http://") && (s.EndsWith("html")||s.EndsWith("/"))))
            }
```

```
type Crawler() =
    let event = new Event<_>()
    let rec CrawlerAgent =
        MailboxProcessor.Start(fun inbox ->
        let rec loop() = async {
            let! url = inbox.Receive()
            let! (cont,urls) = Crawl url
            event.Trigger (url,cont)
            urls |> Seq.iter (fun x->CrawlerAgent.Post x)
            if (not Async.DefaultCancellationToken.IsCancellationRequested)
            then return! loop()
        }
        loop()
        )
    member this.StartIndexing(url) = CrawlerAgent.Post(url)
    member this.StopIndexing() = Async.CancelDefaultToken()
    member this.NewFeed = event.Publish
```

Событийная работа с NUI-контроллерами

- NUI-контроллеры Kinect, Leap Motion – выдают информацию по кадрам, 30fps

```
type LeapMotion() =
    let event = new Event<_>()
    let cntr = new Leap.Controller()
    let rec loop = async {
                    let f = cntr.Frame()
                    post (fun()->event.Trigger(f))
                    do! Async.Sleep(50)
                    if (not IsCancellationRequested) then return! loop
            }
    member this.EventFeed = event.Publish
    member this.Start() = Async.Start loop
    member this.Stop() = Async.CancelDefaultToken()
```

Использование в реальной жизни

- Совместный проект Российского государственного университета физической культуры, спорта, молодёжи и туризма и лаборатории MAILabs
- Оценка показателей здоровья и физического развития детей при помощи Kinect

Переходим к самому интересному!

Программирование роботов

Реактивное программирование в .NET: Reactive Extensions

- IObserver<T>
  - void OnNext(T)
  - void OnError(Exception exn)
  - void OnCompleted()
- IObservable<T>
  - IDisposable Subscribe(IObserver O)
- В Observable можно оборачивать списки (IEnumerable), события и другие привычные объекты

Платформа: ТРИК

- http://trikset.com
- Разработка на базе кафедры системного программирования СПбГУ
- Мощный контроллер, допускающий .NET Framework, WiFi
- Обвязка на базе стандартного конструктора
- Полный стек технологий для обучения:
  - Визуальная среда программирования
  - .NET (C#, F#)
  - Низкий уровень (C)

Программируем робота на F#

```
use model = new Model()
let motorL = model.Motor.["JM1"]
let motorR = model.Motor.["JM2"]
let sensor = model.AnalogSensor[“JA1”]
```

```
motorR.SetPower(20)
motorL.SetPower(-20)
```

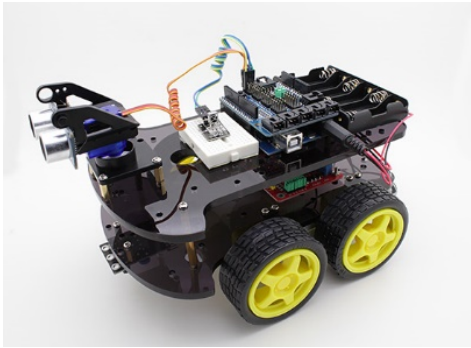
```
let dist x =
    if x<20 || x>80 then 0
    else
        if x<40 then 40
        elif x>60 then -40
        else 0

let power = sensor.ToObservable().Select(dist).DistinctUntilChanged()
use l = power.Subscribe(motorL)
use r = power.Subscribe(motorR)
```

Распределенное реактивное управление роботом

Leap Motion

LeapMotion|> … process …|> TcpClient

Pad |> …. |> Observable.Subscribe(motors)



Управляем роботом через Leap Motion

Робот

```
 let PadInput = Model.Pad.Pads
                     |> Observable.map snd
                     |> Observable.choose (fun x->x)

    use s1=(PadInput |> Observable.map fst).Subscribe(leftWheel)
    use s2=(PadInput |> Observable.map snd).Subscribe(rightWheel)
```

Клиент

```
let p = new TrikPad("192.168.137.86",3333)

let LP = new LeapMotion()
LP.EventFeed
  |> Event.map (fun f -> f.Hands.Rightmost)
  |> Skip 2
  |> Event.filter (fun h -> h.Confidence>0.2f)
  |> Event.map HandAngles
  |> Event.map Control
  |> Tee (printfn "%A")
  |> Event.add (uncurry(p.SendPad 1))
LP.Start()
LP.Stop()
```

```
let HandAngles (hand:Hand) =
(hand.Direction.Yaw,
 hand.Direction.Pitch,hand.PalmNormal.Roll)

let trim x a b =
   if x<a then a else if x>b then b else x

let Control (y,p,r) =
    if p < -0.05f then
       let x = trim (r*1.5f) -1.0f 1.0f
       let spd = (-p+0.05f)*100.0f
       ((int)(spd*(1.0f-x)),
        (int)(spd*(1.0f+x)))
    else (0,0)
```

Управляем роботом через команды из твиттера

Клиент

```
let p = new TrikPad("192.168.1.2",3333)
let send (x,y) =
    p.SendPad 1 x y

let Do x = x |> ParseCommand
             |> List.map Tupelize
             |> List.iter (send)

Do "go forward 10 then stop"

let CmdFeed = new
          TwitterFilterStream(“#robocontrol")

CmdFeed.NewFeed
 |> Event.map (fun x->x.text)
 |> Event.map (fun x->x.Replace("#robo", "")
 |> Event.add (Do)
CmdFeed.StartListening()
```

Робот

```
model.Pad.Pads
        |> Observable.map snd
        |> Observable.choose (fun x->x)
        |> Observable.map Detupleize
        |> Observable.subscribe(
            function
             | Forward v ->
                   setWheels(moveSpeed,moveSpeed)
                   delay (v*delitem)
                   setWheels(0,0)
             | Right ->
                setWheels (-moveSpeed,moveSpeed)
                delay del
                setWheels (0,0)
             | Stop -> setWheels(0,0)
            ) |> ignore
```

Мораль:

```
List |> List.map (fun x->x*2) |> List.filter (fun x -> x>0)
```

```
Sequence |> Seq.map (fun x->x*2) |> Seq.filter (fun x -> x>0)
```

```
EventSource|> Event.map (fun x->x*2) |> Event.filter (fun x -> x>0)
  |> Event.add (fun x-> …)
```

F# - высокодекларативный язык

При одинаковом синтаксисе процесс вычисления может сильно отличаться

Это позволяет эффективно обрабатывать большие данные

В том числе в модели реактивного программирования

При этом:

F# есть на облаке Windows Azure

F# есть на клиенте (Kinect, Leap Motion, W8/WP)

F# - open source

F# имеет сильное сообщество (http://fsharp.org)

F# почти не проприетарный

F# используется на практике (даже в России)

F# используется в преподавании в вузах России (МФТИ, НИУ ВШЭ, СПбГУ, НГУ, МАИ, …)

F# - прекрасен!

Используйте F#!

Дмитрий Сошников

dmitri@soshnikov.com
facebook.com/shwars
twitter.com/shwars
vk.com/shwars
blog.soshnikov.com