CS314. Функциональное программирование

Лекция 19. Разработка веб-приложений. Параллельное программирование

В. Н. Брагилевский

Направление «Фундаментальная информатика и информационные технологии»
Институт математики, механики и компьютерных наук имени И. И. Воровича
Южный федеральный университет

5 декабря 2016 г.

Содержание

1. Разработка веб-приложений и Yesod
2. Параллельное программирование

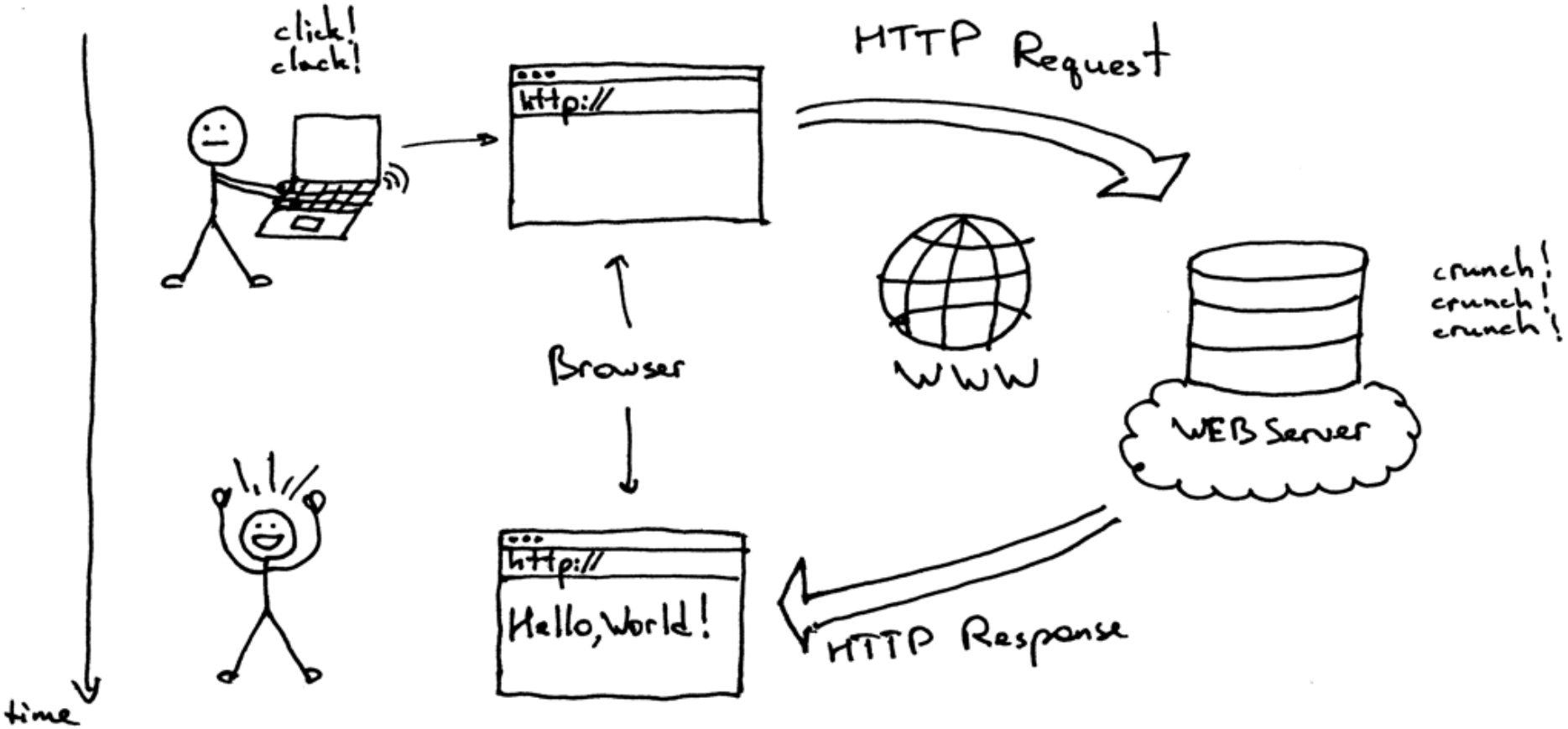
Содержание

1. Разработка веб-приложений и Yesod
   - Служба WWW и веб-приложения
   - Yesod HelloWorld
2. Параллельное программирование

Содержание

1. Разработка веб-приложений и Yesod
   - Служба WWW и веб-приложения
   - Yesod HelloWorld
2. Параллельное программирование

Служба WWW



https://ruslanspivak.com/lsbaws-part1/LSBAWS_HTTP_request_response.png

Компоненты службы WWW

Сторона клиента

- Браузер
- Отображение страниц на основе HTML и CSS
- Исполнение кода на Javascript

Протокол HTTP

- Запросы (requests): GET, PUT и др.
- Ответы (responses): HTML + CSS + Javascript + всё что-угодно.
- Протокол HTTP не поддерживает состояние (HTTP is stateless).

Сторона сервера

- Веб-сервер (Apache, Nginx, Warp)
- Генерация ответов специальным приложением (движком)
- Файловая система сервера и СУБД

Основные задачи серверных веб-приложений

- Маршрутизация запросов (routing)
- Обработка запросов (handling) и бизнес-логика
- Шаблонизаторы (templates) и компоненты (widgets)
- Поддержка сессий (хранение пользовательского состояния)
- Доступ к внешним данным (СУБД)

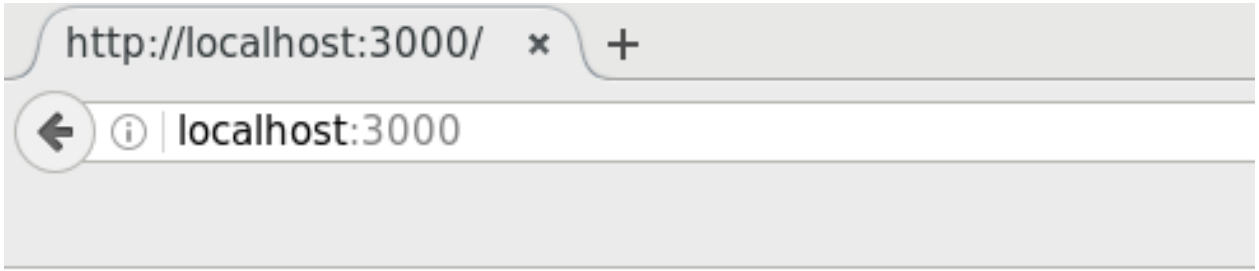
Содержание

1. Разработка веб-приложений и Yesod
   - Служба WWW и веб-приложения
   - Yesod HelloWorld
2. Параллельное программирование

```
[hello]$ ls
hello.cabal  HelloWorld.hs  stack.yaml
[hello]$ cat hello.cabal
name:                hello
version:             0.0.0
cabal-version:       >= 1.8
build-type:          Simple

executable           HelloWorld
  main-is:           HelloWorld.hs

  build-depends:  base
                , yesod-core
[hello]$ cat stack.yaml
resolver: lts-6.26
[hello]$  stack setup
stack will use a sandboxed GHC it installed
...
```

```haskell
{-# LANGUAGE OverloadedStrings     #-}
{-# LANGUAGE QuasiQuotes           #-}
{-# LANGUAGE TemplateHaskell       #-}
{-# LANGUAGE TypeFamilies          #-}

import Yesod

data HelloWorld = HelloWorld

mkYesod "HelloWorld" [parseRoutes|
/ HomeR GET
|]

instance Yesod HelloWorld

getHomeR :: Handler Html
getHomeR = defaultLayout [whamlet|<h1>Hello World!|]

main :: IO ()
main = warp 3000 HelloWorld
```

```
[hello]$ stack runghc HelloWorld.hs
05/Dec/2016:09:21:57 +0300 [Info#yesod-core] Application
 launched @(yesod_75W0RKv384xIY4MKLHknpM:Yesod.Core.Dispatch
    ./Yesod/Core/Dispatch.hs:157:11)
127.0.0.1 - - [05/Dec/2016:09:22:35 +0300] "GET / HTTP/1.1"
    200 97 "" "Mozilla/5.0 (X11; Linux x86_64; rv:50.0)
    Gecko/20100101 Firefox/50.0"
```



Hello World!

```
<!DOCTYPE html>
<html><head><title></title></head><body><h1>Hello World!</h1>
</body></html>
```

```haskell
{-# LANGUAGE OverloadedStrings  #-}
{-# LANGUAGE QuasiQuotes        #-}
{-# LANGUAGE TemplateHaskell    #-}
{-# LANGUAGE TypeFamilies       #-}

import Yesod

data HelloWorld = HelloWorld

mkYesod "HelloWorld" [parseRoutes|
/ HomeR GET
|]

instance Yesod HelloWorld

getHomeR :: Handler Html
getHomeR = defaultLayout [whamlet|<h1>Hello World!|]

main :: IO ()
main = warp 3000 HelloWorld
```

Расширения GHC

Основной тип приложения (Foundation)

Ресурсы приложения

Обработчик и шаблон ответа

Веб-сервер

Маршрутизация запросов

```
mkYesod "HelloWorld" [parseRoutes|
/ HomeR GET
|]
```

Сгенерированный код

```
instance RenderRoute HelloWorld where
    data Route HelloWorld = HomeR
        deriving (Show, Eq, Read)
    renderRoute HomeR = ([], [])

instance ParseRoute HelloWorld where
    parseRoute ([], _) = Just HomeR
    parseRoute _       = Nothing
```

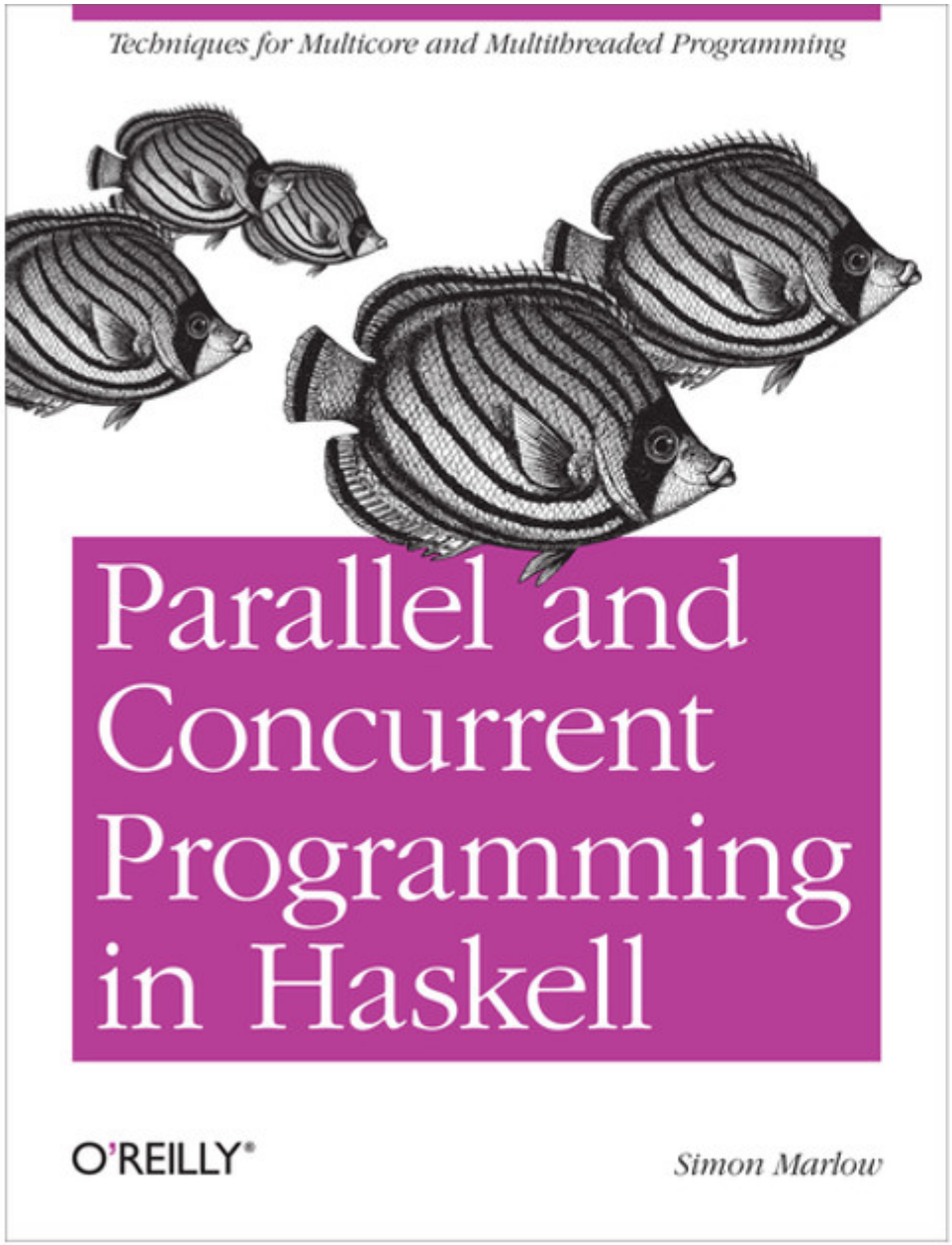
Сгенерированный код

```
instance YesodDispatch HelloWorld where
    yesodDispatch env req =
        yesodRunner handler env mroute req
      where
        mroute = parseRoute (pathInfo req, textQueryString req)
        handler =
            case mroute of
                Nothing -> notFound
                Just HomeR ->
                    case requestMethod req of
                        "GET" -> getHomeR
                        _     -> badMethod

type Handler = HandlerT HelloWorld IO
```

```
getHomeR :: Handler Html
getHomeR = defaultLayout [whamlet|<h1>Hello World!|]
```

Содержание

1. Разработка веб-приложений и Yesod
2. Параллельное программирование
   - Монада Eval и стратегии вычислений


Techniques for Multicore and Multithreaded Programming
Parallel and Concurrent Programming in Haskell
O'REILLY
Simon Marlow


Функциональное программирование
Саймон Марлоу
Параллельное и конкурентное программирование
на языке Haskell
ДМК
издательство
OZON.RU

Терминология

- Параллелизм и конкурентность
- Эффективность и модульность
- Детерминированная и недетерминированная модели программирования

Параллельное программирование в Haskell

- Декларативность
- Деление задачи на параллельно исполняемые фрагменты
- Степень детализации
- Зависимости по данным
- Детерминированность результатов

Содержание

1 Разработка веб-приложений и Yesod

2 Параллельное программирование
- Монада Eval и стратегии вычислений

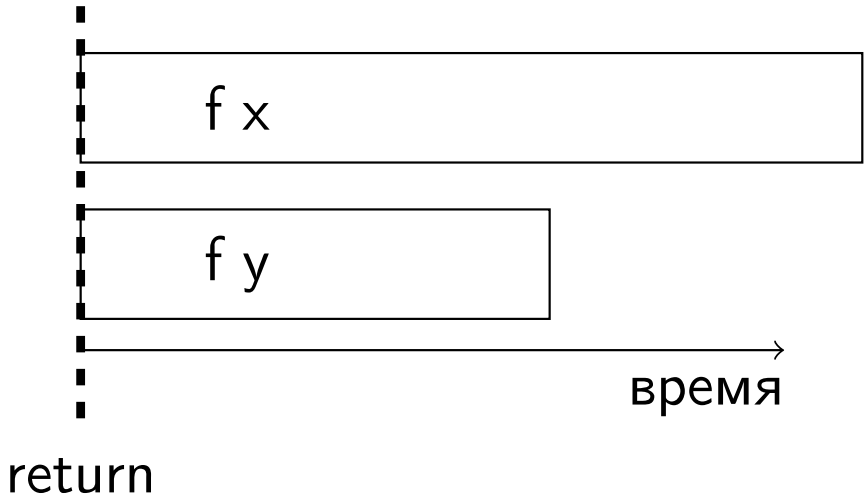
Монада Eval

```haskell
import Control.Parallel.Strategies

data Eval a

instance Monad Eval

runEval :: Eval a -> a
rpar :: a -> Eval a
rseq :: a -> Eval a
```

- Вычисление до слабой головной нормальной формы.

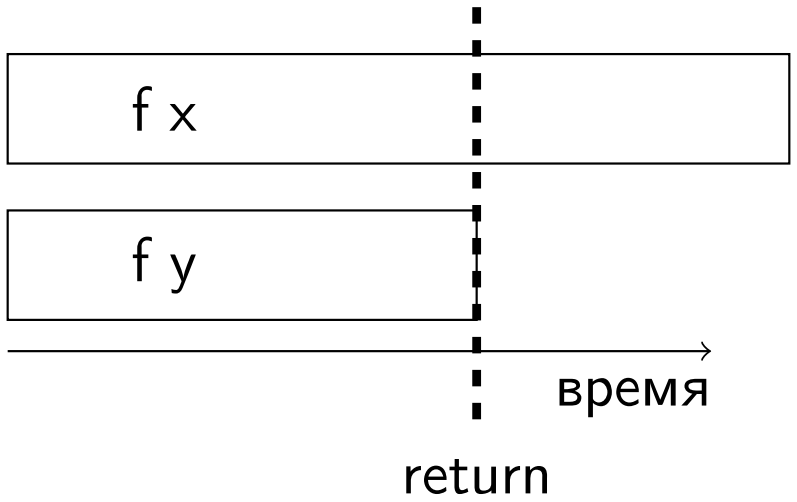
Пример: rpar/rpar

```
runEval $ do
    a <- rpar (f x)
    b <- rpar (f y)
    return (a,b)
```



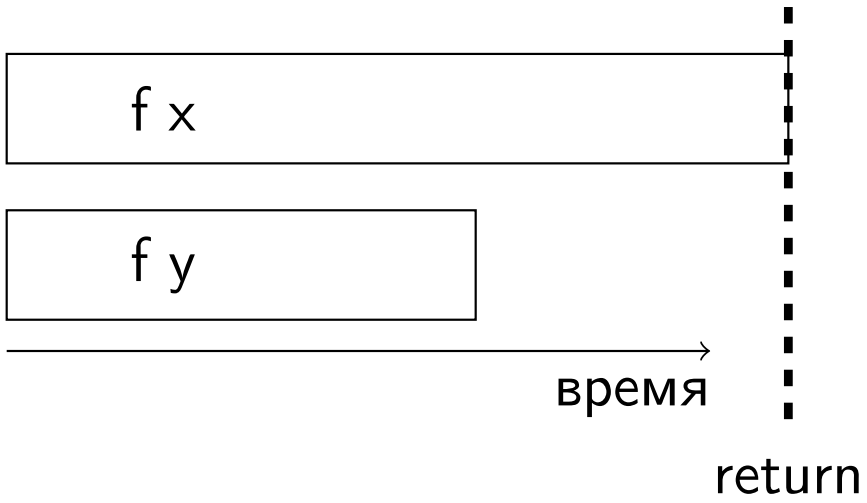
Пример: rpar/rseq

```
runEval $ do
    a <- rpar (f x)
    b <- rseq (f y)
    return (a,b)
```



Пример: rpar/rseq/rseq

```
runEval $ do
    a <- rpar (f x)
    b <- rseq (f y)
    rseq a
    return (a,b)
```



Пример: rpar/rpar/rseq/rseq

```
runEval $ do
    a <- rpar (f x)
    b <- rpar (f y)
    rseq a
    rseq b
    return (a,b)
```

```haskell
runEval $ do
    xs' <- rpar (map f xs)
    ys' <- rpar (map f ys)
    rseq xs'
    rseq ys'
    return (xs'++ys')
```

Вычисление до слабой головной нормальной формы!

```haskell
runEval $ do
    xs' <- rpar (force $ map f xs)
    ys' <- rpar (force $ map f ys)
    rseq xs'
    rseq ys'
    return (xs'++ys')
```

```haskell
import Control.DeepSeq

force :: NFData a => a -> a
```

- Просто?
- Слишком просто для языка Haskell!
- Необходимо отделить вычисление от способа распараллеливания (нужна новая абстракция!).
- Стратегии вычисления:

```
type Strategy a = a -> Eval a
```

Пример стратегии

```haskell
parPair :: Strategy (a,b)
parPair (a,b) = do
  a' <- rpar a
  b' <- rpar b
  return (a',b')
```

Вспомогательная функция

```haskell
using :: a -> Strategy a -> a
x `using` s = runEval (s x)
```

```haskell
(fib 35, fib 36) `using` parPair
```

Параметризованные стратегии

```
evalPair :: Strategy a -> Strategy b -> Strategy (a,b)
evalPair sa sb (a,b) = do
  a' <- sa a
  b' <- sb b
  return (a',b')
```

```
parPair :: Strategy (a,b)
parPair = evalPair rpar rpar
```

```
rpar :: a -> Eval a
rseq :: a -> Eval a
```

Комбинирование стратегий

```
rdeepseq :: NFData a => Strategy a
rdeepseq x = rseq (force x)
```

- Добавляем rpar:

```
parPair :: Strategy a -> Strategy b -> Strategy (a,b)
parPair sa sb = evalPair (rparWith sa) (rparWith sb)
```

```
parPair rdeepseq rdeepseq :: (NFData a, NFData b)
                          => Strategy (a,b)
```

Частично параллельные вычисления

```
r0 :: Strategy a
r0 x = return x

evalPair (evalPair rpar r0) (evalPair rpar r0)
                          :: Strategy ((a,b),(c,d))
```

Пример: функция parMap

- Алгоритм: map
- Параллелизм: параллельное вычисление элементов списка

```
parMap :: (a -> b) -> [a] -> [b]
parMap f xs = map f xs `using` parList rseq

evalList :: Strategy a -> Strategy [a]
evalList strat []       =   return []
evalList strat (x:xs)   =   do
  x'    <- strat x
  xs'   <- evalList strat xs
  return (x':xs')

parList :: Strategy a -> Strategy [a]
parList strat = evalList (rparWith strat)
```

Модуль Control.Parallel.Strategies

```
type Strategy a = a -> Eval a

using :: a -> Strategy a -> a
dot :: Strategy a -> Strategy a -> Strategy a
r0 :: Strategy a
rseq :: Strategy a
rdeepseq :: NFData a => Strategy a
rpar :: a -> Eval a
rparWith :: Strategy a -> Strategy a
evalList :: Strategy a -> Strategy [a]
parList :: Strategy a -> Strategy [a]
parMap :: Strategy b -> (a -> b) -> [a] -> [b]
...
```

Монада Eval: заключение

- Ленивые структуры данных (пара, список. . .).
- Декларативное описание способа распараллеливания и его применение.
- Проблемы со степенью детализации.

Библиография

1. Yesod Book
   `http://www.yesodweb.com/book/`
2. С. Марлоу. Параллельное и конкурентное программирование на языке Haskell. Текст на английском доступен онлайн: `http://chimera.labs.oreilly.com/books/1230000000929/index.html`