OCaml — урок 5

Обработка ошибок

Скворцов В. С., Голузин Е. К., Горюнов М. Ю.

Обработка ошибок (1/8)

В OCaml есть три способа избежать игнорирования ошибок:

1. **Исключения (exceptions)**
2. **Option**
3. **Result**

Обработка ошибок (2/8)

Exceptions:

```
exception Foo of string

let i_will_fail () =
  raise (Foo "Cringe")

i_will_fail ()
(* Exception: Foo "Cringe". *)
```

try ... with ...:

```
try i_will_fail () with Foo _ -> ()
(* unit = () *)
```

Обработка ошибок (3/8)

Предопределенные исключения:

```
1 / 0 (*Exception: Division_by_zero *)

List.find (fun x -> x mod 2 = 0) [1; 3; 5] (* Exception: Not_found *)

String.sub "Hello world!" 3 (-2)
(* Exception: Invalid_argument "String.sub / Bytes.sub" *)

let rec loop x = x :: loop x
loop 42 (* Stack overflow during evaluation (looping recursion?) *)

(* Other predefined exceptions:
Exit, Not_found, Invalid_argument, Failure *)
```

Обработка ошибок (4/8)

Fun.protect:

```
let head_channel chan =
  let rec loop acc n = match input_line chan with
    | line when n > 0 -> loop (line :: acc) (n - 1)
    | _ -> List.rev acc in
  loop []

let head_file filename n =
  let ic = open_in filename in
  let finally () = close_in ic in
  let work () = head_channel ic n in
  Fun.protect ~finally work
```

Обработка ошибок (5/8)

Stack trace:

```
$ OCAMLRUNPARAM=b ./myprogram
```

или

```
let () =
  Printexc.record_backtrace true
```

Printing:

```
let notify_user f =
  try f () with e ->
    let msg = Printexc.to_string e
    and stack =
Printexc.get_backtrace () in
    Printf.eprintf stack;

      raise e
```

Обработка ошибок (6/8)

Option:

```
let div_opt m n =
  try Some (m / n) with
    Division_by_zero -> None

let find_opt p l =
  try Some (List.find p l) with
    Not_found -> None
```

Обработка ошибок (7/8)

Result:

```
let int_of_string_result s =
  try
    Ok (int_of_string s)
  with
  | Failure _ -> Error "Invalid integer"

let increment_if_valid s =
  Result.map (fun x -> x + 1) (int_of_string_result s)

let handle_error s =
  Result.map_error (fun e -> "Error: " ^ e) (int_of_string_result s)
```

Обработка ошибок (8/8)

Result:

```
let () =
  let test_strings = ["123"; "abc"; "456"] in
  List.iter (fun s ->
  match increment_if_valid s with
    | Ok n -> Printf.printf "Incremented value: %d\n" n
    | Error e -> Printf.printf "Failed to increment: %s\n" e
  ) test_strings;

  List.iter (fun s ->
    match handle_error s with
    | Ok n -> Printf.printf "Parsed value: %d\n" n
    | Error e -> Printf.printf "%s\n" e
  ) test_strings
```

Управление зависимостями (1/2)

```
$opam install . --deps-only --with-test –with-doc

$opam switch set <switch_name>
$opam install . --deps-only --with-test –with-doc

$ opam exec -- dune build
```

Управление зависимостями (2/2)

.opam file:

```
opam-version: "2.0"
synopsis: "A short, but powerful statement about your project"
description: "An complete and exhaustive description everything your
project does."
depends: [
  "ocaml" {>= "4.08.0"}
  "dune"
  "alcotest" {with-test}
  "odoc" {with-doc}
]
```

Последовательности (1/2)

```
let rec ints n : int Seq.t = fun () -> Seq.Cons (n, ints (n + 1));;

Seq.iter print_int (ints 0)

Seq.ints 0 |> Seq.take 5 |> List.of_seq (* [0; 1; 2; 3; 4] *)
```

Последовательности (1/2)

```
let rec trial_div seq () = match seq () with
  | Seq.Cons (m, seq) -> Seq.Cons (m, trial_div (Seq.filter (fun n -> n
mod m > 0) seq))
  | seq -> seq

let primes = Seq.ints 2 |> trial_div



primes |> Seq.take 5 |> List.of_seq (* [2; 3; 5; 7; 11]  *)
```

Хэш-таблицы (1/2)

```
let my_hash = Hashtbl.create 123456

Hashtbl.add my_hash "h" "hello"
Hashtbl.add my_hash "h" "hi"
Hashtbl.add my_hash "w" "world"
Hashtbl.add my_hash "w" "wine"

Hashtbl.find my_hash "h" (* "hi" *)
Hashtbl.find_all my_hash "h" (* ["hello"; "hi"] *)

Hashtbl.remove my_hash "h"

Hashtbl.find my_hash "h" (* «hi" *)
```

Хэш-таблицы (2/2)

```
Hashtbl.replace my_hash "t" "try"
Hashtbl.replace my_hash "t" "test"
Hashtbl.find_all my_hash "t" (* ["test"] *)

Hashtbl.remove my_hash "t" (* Exception: Not_found *)

Hashtbl.mem my_hash "h" (* true *)
```

Sets (1/2)

```
module StringSet = Set.Make(String)

StringSet.empty (* StringSet.t = <abstr> *)
StringSet.empty |> StringSet.to_list (* [] *)

StringSet.singleton "hello" (* StringSet.t = <abstr> *)
StringSet.(singleton "hello" |> to_list) (* ["hello"] *)

StringSet.of_list ["hello"; "hi"] (* StringSet.t = <abstr> *)
StringSet.(of_list ["hello"; "hi"] |> to_list) (* ["hello"; «hi"] *)
```

Sets (2/2)

```
let first_set = ["hello"; "hi"] |> StringSet.of_list
let second_set = ["good morning"; "hi"] |> StringSet.of_list

StringSet.(first_set |> add "good morning" |> to_list)
(* ["good morning"; "hello"; "hi"] *)

StringSet.(first_set |> remove "hello" |> to_list) (* ["hi"] *)

StringSet.(union first_set second_set |> to_list)
(* ["good morning"; "hello"; "hi"] *)

StringSet.(inter first_set second_set |> to_list) (* ["hi"] *)

StringSet.(diff first_set second_set |> to_list) (* hello *)
```

Спасибо за интерес