Peter the Great
Saint-Petersburg Polytechnic University

Ocaml: модули и интерфейсы

21/11/2025

Презентацию подготовили:
- Дрекалов Никита
- Соколов Артём

POLYTECH

План презентации

- Основы модулей
  - `Общая информация и пример простого модуля`
- Сигнатуры и .mli файлы
  - `Интерфейсы и сигнатуры`
- Расширенные возможности модулей
  - `Вложенные модули, функторы, директива include`

Основы модулей

Общая информация

- `Модуль` в `OCaml` — это пространство имён, которое может содержать:
  - Типы
  - Значения (функции, константы),
  - Подмодули,
  - Исключения

Пример простого модуля

- Для объявления модуля используется ключевое слово `module`

```
module Math = struct
  let add x y = x + y
  let square x = x * x
end

let result = Math.add 2 3
```

Файлы .ml = модули

- Каждый файл <name>.ml автоматически порождает модуль <name>
- В math.ml:

```
let add x y = x + y
let square x = x * x
```

- В другом файле:

```
let result = math.add 1 2
```

Сигнатуры и .mli файлы

Интерфейсы: .mli файлы

Каждому .ml файлу может соответствовать .mli интерфейс

Интерфейс определяет:

- Что именно модуль экспортирует
- Какие типы и значения доступны извне
- Обеспечивает скрытие реализации (абстракцию)

```
let add : float float -> float
let square : float -> float
```

Сигнатуры – объявление

```
module type STACK = sig
  type 'a t  (* абстрактный тип - реализация скрыта *)

  val create : unit -> 'a t
  val push : 'a t -> 'a -> unit
  val pop : 'a t -> 'a option
  val is_empty : 'a t -> bool
end
```

Сигнатуры – реализация

```
module Stack : STACK = struct
  type 'a t = 'a list ref  (* реализация скрыта от пользователя *)

  let create () = ref []
  let push stack x = stack := x :: !stack
  let pop stack =
    match !stack with
    | [] -> None
    | x :: xs -> stack := xs; Some x
  let is_empty stack = !stack = []
end
```

Модуль `Stack` реализует сигнатуру, но тип `'a t` остается абстрактным для внешнего мира

Расширенные возможности модулей

Вложенные модули

```
module Container = struct
  module Stack = struct
    type 'a t = 'a list
    let empty = []
  end

  module Queue = struct
    type 'a t = 'a list * 'a list
    let empty = ([], [])
  end
end
```

Функторы (модули высшего порядка)

```
module type ORDERED = sig
  type t
  val compare : t -> t -> int
end

module MakeSet (Elt : ORDERED) = struct
  type elt = Elt.t
  type t = elt list
  let empty = []
  let add x set =
    if List.exists (fun y -> Elt.compare x y = 0)
set
    then set
    else x :: set
end

module IntSet = MakeSet(struct
  type t = int
  let compare = compare
end)
```

- Функторы – это функции, принимающие модули и возвращающие модули

Include и открытие модулей

- Директива `include` включает все определения из одного модуля в другой

```
module BaseStack = struct
  type 'a t = 'a list
  let empty = []
end

module Stack = struct
  include BaseStack
  let push x stack = x :: stack
  let pop = function
    | [] -> None
    | x :: xs -> Some (x, xs)
end
```

Спасибо за внимание!