Reasonml

- Reasonml 简介
- 数据类型
- 函数
- 模式匹配
- 模块
- 如何与 JavaScript 交互
- 类型推断
- 如何搭建前端项目（使用 ReasonReact）
- 优缺点
- 资源

Reasonml(OCaml)

https://reasonml.github.io/

ReasonML 是在 Facebook 创建的一种新的函数式静态类型编程语言。实质上，它是编程语言 OCaml 的新 C 语言语法。新的语法旨在与 JavaScript 进行互操作，并且更容易被 JavaScript 程序员接受。此外，它消除了 OCaml 语法（古怪的语法）的特性。ReasonML 还支持 JSX（ Facebook 的 React 框架使用的 JavaScript 内部 HTML 模板的语法）。 ReasonML 基于 OCaml。Reasonml 可以用于开发前端项目

简单来说
Reasonml 是

- 一种编写 React 应用程序的新方法;
- OCaml 语义且 JavaScript 语法友好;
- 静态类型 – 带有类型推断;
- 函数式，但不是纯粹的函数式;
- 主要编译为 JavaScript （通过BuckleScript）;
- 由 Facebook 和 Bloomberg 创建。

主要作者

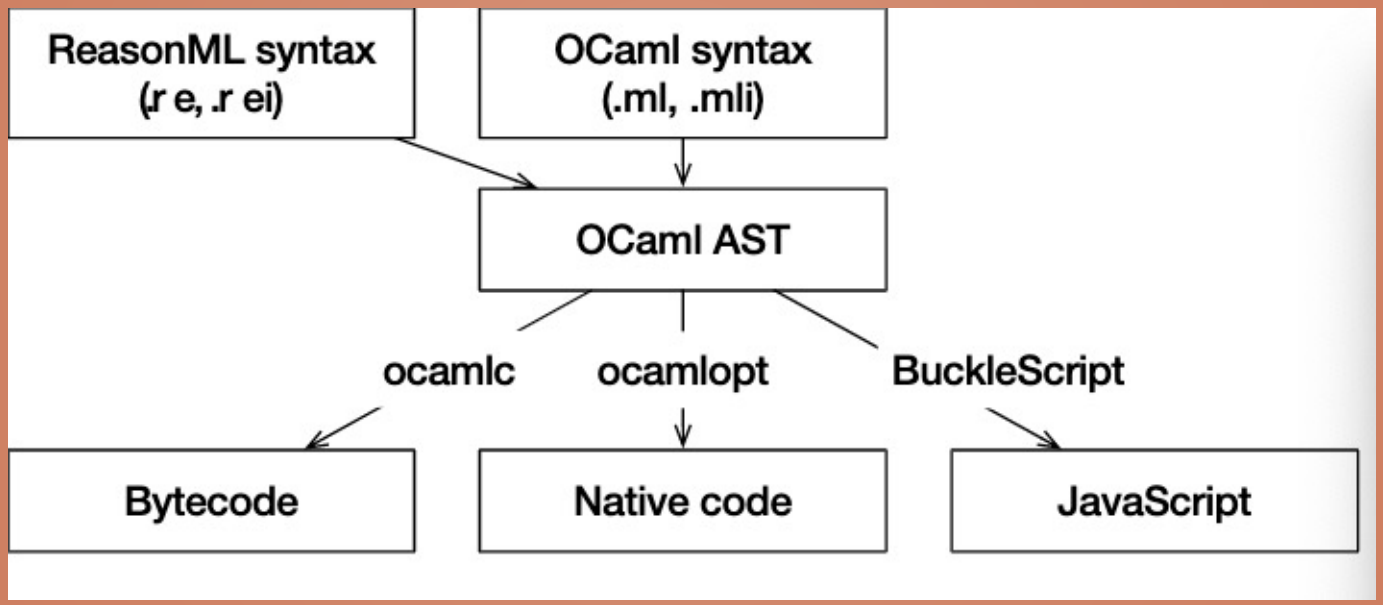
jordwalke （ Reactjs 创建者）

https://github.com/jordwalke

Hongbo Zhang（BuckleScript 编译器作者）

https://github.com/bobzhang

下图显示了 ReasonML 如何使用 OCaml 生态系统和Javascript 生态。



数据类型

字符串	**"Hello"**
字符	'x'
整型数字	23, -23
浮点型数字	23.0, -23.0
整型数字加法	23 + 1
浮点型数字加法	23.0 +. 1.0
整型数字除法/乘法	2 / 23 * 1
浮点型数字除法/乘法	2.0 /. 23.0 *. 1.0
浮点型数字求幂	2.0 ** 2.0
字符串组合	"Hello " ++ "World"
比较运算符	>, <, >=, =<
布尔运算符	! && ||
引用（浅）比较，结构（深）比较	===, ==
列表(不可变)	[1, 2, 3]
数组	[|1, 2, 3|]
元组（Tuple）	(1, "string")
记录（Records）	type player = {score: int}; {score: 100}
对象	type tesla = {var red = "red"; pub color = red;}; tesla#color

记录类型

```
type person = {
 name: string,
 age: int,
};
let jay = {name: "jay", age: 100};
```

获取记录字段

```
jay.name; /* "jay" */
jay..age; /* 100 */
```

变体类型

```
type shoesColor =
 | Red
 | Blue
 | Green
 | Black;
```

数据结构变体类型

```
type point =
 | Point(float, float);
type shape =
 | Rectangle(point, point)
 | Circle(point, float);
```

自递归变体类型

```
type intTree =
 | Empty
 | Node(int, intTree, intTree);
```

多态类型

```
type list('a) =
 | Nil
 | Cons('a, list(‘a));

let myList = Cons("foo", Cons("bar", Cons("baz", Nil)));
```

Option 类型

```
type option('a) =
 | None
 | Some('a);
```

```
let division = (a: int, b: int) : option(int) =>
 if (b === 0) {
 None;
 } else {
 Some(a / b);
 };
let myLuckyNumber = division(23, 45);
Js.log(myLuckyNumber)
```

模式匹配

switch

```
let min = (x, y) =>
 if (x < y) {
   x;
 } else {
   y;
 };

let min = (x, y) => switch(x < y) {
 | true => x
 | false => y
};
```

switch

```
let tuple = (true, true);

let result = switch tuple {
    | (false, false) => false
    | (false, true) => false
    | (true, false) => false
    | (true, true) => true
};
/* result == true */

/* 通过下划线下划线和变量来简化 */

let result = switch tuple {
    | (false, _) => false
    | (true, x) => x
};
/* result == true */
```

通过 let 做模式匹配

```
let tuple = (7, 4);

let (x, y) =  tuple;

Js.log(x); /* 7 */

Js.log(y);/* 4 */
```

函数

普通函数

```
let add = (a, b) => a + b;
let alsoAdd = a => b => a + b;
let add = (x, y) => x + y;
let inc = add(1);
inc(10); /* 11 */

 /* 标签参数函数 */
let add = (~x, ~y) => {x + y};
add(~y=5, ~x=6); // 标签参数位置可以随便定义
```

递归函数

```
// 阶乘 if else 实现
let rec factorial = (x) =>
    if(x <= 0) {
        1;
    } else {
        x * factorial(x – 1);
    };
// 阶乘模式匹配实现
let rec factorial = (x: int) : int =>
    switch (x) {
    | 0 => 1
    | x => x * factorial(x – 1)
    };
```

https://reasonml.github.io/en/try.html?

rrjsx=true&reason=DYUwLgBATiDGEDMCGswHsoEsnAgXggAoAPALgkwDswBKCcqyPAPgCgIOIBnAd0zFgALIsToBvdpwA+EAAz5m

EAIySOM4gogaAVIhTosOEhAC0ymqogBfANytWAKS4A6YGgDmhZKgzZghJVkaGiA

模块

在 Reasonml 中文件即模块（即一个文件就相当一个模块），例如：

```
// Calc.re

let add = (x, y) => x + y;

// index.re

open Calc

let add1 = add(1, 2)
```

也可以通过如下这种方式在一个文件中定义

```
// index.re

module Calc = {
 let add = (x, y) => x + y;
};
Calc.add(4, 5);
```

https://reasonml.github.io/en/try.html?

rrjsx=true&reason=LYewJgrgNgpgBAYQIZQMZwLxwN4Cg6wAucSYYmcAFAB4A0cAngJSYB

8c1cA1IwNy4BffsjQA6UmEoAWegFYm-XACkAzqKggA5pRGpxZaXKZMgA

与 Javascript 交互

通过 bucklescript 语法

```
let add:(int, int) => int = [%raw {|
  function(a, b) {
    console.log("hello from raw JavaScript!");
    return a + b
  }
|}];
```

%raw 为 bucklescript 语法

https://bucklescript.github.io/docs/zh-CN/embed-raw-javascript

使用浏览器中 Javascript encodeURI 方法

```
[@bs.val] external encodeURI: string => string = "encodeURI";
let result = encodeURI("https://reasonml.github.io");

Js.log(result)
```

更多用法请查看
https://bucklescript.github.io/docs/zh-CN/interop-cheatsheet 和 bucklescript详细文档

类型推断

```
let add = (x, y) => {
  x + y;
}

Js.log(add(4,5))

// Js.log(add(4,"5")) 这里会类型错误
```

https://reasonml.github.io/en/try.html?

rrjsx=true&reason=DYUwLgBAhgJjEF4IAoAeAaCBPAIlgfBAN4BQEEqEA1NgNwkC+JJAUgM4B0wA9gObKwYyACzoArDhzMA9ACoI7LnwF

wR6AERj1kiIE34wDOJgLHIA3j6Bo9UCYqYHvoiLOmtOPfoLWbtOIA

优缺点

优点：

- 类型推断，不需要强制定义类型
- 编译时就能发现类型不匹配的错误，编译器可以帮助我们提前避免程序在运行期间有可能发生的一些错误

缺点：

- 学习成本高，生态不完善

资源列表

Reason-react：https://reasonml.github.io/reason-react/

Reasonml：https://reasonml.github.io/en/

Bucklescript：https://bucklescript.github.io/

安装 reason-react: https://reasonml.github.io/reason-react/docs/en/installation.html

谢谢！