pontus nagy

@p1xelHer0

an introduction to ReasonReact

React

- Components
- Render
- Children
- Props
- State

Components

Component

```
/* Component.re */
let component = ReasonReact.statelessComponent("Component");

let make = _children => {
  ...component,
  render: _self => <div />,
};


/* Index.re */
ReactDOMRe.renderToElementWithId(<Component />, "index");
```

React

- Components ✅
- render
- Children
- Props
- State

render

render

```
/* Component.re */
let component = ReasonReact.statelessComponent("Component");

let make = _children => {
  ...component,
  render: _self => <div />,
};


/* Index.re */
ReactDOMRe.renderToElementWithId(<Component />, "index");
```

render

```
/* render: self -> ReasonReact.reactElement */

render: _self => <div />
```

React

- Components ✅
- render ✅
- Children
- Props
- State

children

children

```
/* Component1.re */
let component = ReasonReact.statelessComponent("Component1");

let make = _children => {
  ...component,
  render: _self =>
    <Component2> {ReasonReact.string("children!")} </Component2>,
};
```

```
/* Component2.re */
let component = ReasonReact.statelessComponent("Component2");

let make = children => {
  ...component,
  render: _self => <div> {children} </div>,
};
```

children

```
children: ReasonReact.reactElement
```

follow the types

```
render: self -> ReasonReact.reactElement

children: ReasonReact.reactElement

... -> ReasonReact.reactElement
```

giveth us thy reactElement

```
render: self -> ReasonReact.reactElement

ReasonReact.string: string -> ReasonReact.reactElement
ReasonReact.array: array('a) -> ReasonReact.reactElement
ReasonReact.null: ReasonReact.reactElement
```

ReasonReact.string

```
/* ReasonReact.string: string -> ReasonReact.reactElement */

render: _self => <div> {ReasonReact.string("Hello world!")} </div>
```

string_of_...

```
/* ReasonReact.string: string -> ReasonReact.reactElement */

render: _self =>
  <div>
    <div> {ReasonReact.string(string_of_int(3))} </div>
    <div> {ReasonReact.string(string_of_float(3.0))} </div>
    <div> {ReasonReact.string(string_of_bool(true))} </div>
  <div>

/* save some keystrokes */
let s = ReasonReact.string
```

ReasonReact.array

```
/* ReasonReact.array:
     array(ReasonReact.reactElement) -> ReasonReact.reactElement */

/* list(int) */
let list = [1, 2, 3];

render: _self =>
  <div>
    {
      /* list(ReasonReact.reactElement) */
      List.map(number =>
        <div> {ReasonReact.string(string_of_int(number))} </div>,
        list,
      )
      /* array(ReasonReact.reactElement) */
      |> Array.of_list
      |> ReasonReact.array
    }
  </div>
```

ReasonReact.array

```
/* ReasonReact.array:
     array(ReasonReact.reactElement) -> ReasonReact.reactElement */

/* array(int) */
let array = [|1, 2, 3|];

render: _self =>
  <div>
    {
      /* array(ReasonReact.reactElement) */
      Array.map(number =>
        <div> {ReasonReact.string(string_of_int(number))} </div>,
        array,
      )
      |> ReasonReact.array
    }
  </div>
```

ReasonReact.null

```
/* ReasonReact.null: ReasonReact.reactElement */

render: _self =>
  <div>
    {
      1 === 2 ?
        <span> {ReasonReact.string("Oops!")} </span> : ReasonReact.null
    }
  </div>
```

ReasonReact.array

```
/* ReasonReact.array:
     array(ReasonReact.reactElement) -> ReasonReact.reactElement */

/* list(int) */
let list = [1, 2, 3];

render: _self =>
  <div>
    {
      /* list(ReasonReact.reactElement) */
      List.map(number =>
        <div> {ReasonReact.string(string_of_int(number))} </div>,
        list,
      )
      /* array(ReasonReact.reactElement) */
      |> Array.of_list
      |> ReasonReact.array
    }
  </div>
```

ReasonReact.array

```
/* ReasonReact.array:
     array(ReasonReact.reactElement) -> ReasonReact.reactElement */

/* list(int) */
let list = [1, 2, 3];

render: _self =>
  {
    /* list(ReasonReact.reactElement) */
    List.map(number =>
      <div> {ReasonReact.string(string_of_int(number))} </div>,
      list,
    )
    /* array(ReasonReact.reactElement) */
    |> Array.of_list
    |> ReasonReact.array
  }
```

React

- Components ✅
- render ✅
- Children ✅
- Props
- State

props

children

```
/* Component1.re */
let component = ReasonReact.statelessComponent("Component1");

let make = _children => {
  ...component,
  render: _self =>
    <Component2> {ReasonReact.string("children!")} </Component2>,
};
```

```
/* Component2.re */
let component = ReasonReact.statelessComponent("Component2");

let make = children => {
  ...component,
  render: _self => <div> {children} </div>,
};
```

props

```
/* Component1.re */
let component = ReasonReact.statelessComponent("Component1");

let make = _children => {
  ...component,
  render: _self =>
    <Component2 message="hi there" />,
};
```

props

```
/* Component1.re */
let component = ReasonReact.statelessComponent("Component1");

let make = _children => {
  ...component,
  render: _self =>
    <Component2 message="hi there" />,
};
```

```
/* Component2.re */
let component = ReasonReact.statelessComponent("Component2");

let make = (~message, _children) => {
  ...component,
  render: _self => <div> {ReasonReact.string(message)} </div>,
};
```

props

```
/* Component1.re */
let component = ReasonReact.statelessComponent("Component1");

let make = _children => {
  let onClick = _event => Js.log("hi there");
  {
    ...component,
    render: _self => <Component2 onClick=onClick />,
  }
};


/* Component2.re */
let component = ReasonReact.statelessComponent("Component2");

let make = (~onClick, _children) => {
  ...component,
  render: _self => <div onClick=onClick />,
};
```

props

```
/* Component1.re */
let component = ReasonReact.statelessComponent("Component1");

let make = _children => {
  let onClick = _event => Js.log("hi there");
  {
    ...component,
    render: _self => <Component2 onClick />,
  }
};


/* Component2.re */
let component = ReasonReact.statelessComponent("Component2");

let make = (~onClick, _children) => {
  ...component,
  render: _self => <div onClick />,
};
```

React

- Components ✅
- render ✅
- Children ✅
- Props ✅
- State

state

state

```
type state = {greeting: string};

let component = ReasonReact.reducerComponent("Component3");

let make = _children => {
  ...component,
  initialState: () => {greeting: "hi"},
  render: self =>
    <div> {ReasonReact.string(self.state.greeting ++ " ReasonSTHLM")} </div>,
};
```

state

```
type state = {greeting: string};

let component = ReasonReact.reducerComponent("Component3");

let make = _children => {
  ...component,
  initialState: () => {greeting: "hi"},
  render: self =>
    <div> {ReasonReact.string(self.state.greeting ++ " ReasonSTHLM")} </div>,
};
```

updating state

```
(action, state) -> state
```

state

```
let make = _children => {
  /* the other stuff is still here! */
  reducer: (_action, _state) => ...?
};
```

action

```
type action =
  | DoIt;

let make = _children => {
  /* the other stuff is still here! */
  reducer: (action, _state) =>
    switch (action) {
    | DoIt => ...?
    },
};
```

action

```
type action =
  | DoIt;

let make = _children => {
  /* the other stuff is still here! */
  reducer: (action, _state) =>
    switch (action) {
    | DoIt => ...?
    },
};
```

```
(action, state) -> state
```

types

```
reducer: ('action, 'state) => update('state, 'retainedProps, 'action)

update('state, 'retainedProps, 'action) =
  | NoUpdate
  | Update('state)
  | SideEffects(self('state, 'retainedProps, 'action) => unit)
  | UpdateWithSideEffects(
      'state,
      self('state, 'retainedProps, 'action) => unit,
    )
```

types

```
reducer: ('action, 'state) => update('state, 'retainedProps, 'action)

update('state, 'retainedProps, 'action) =
  | NoUpdate
  | Update('state)
  | SideEffects(self('state, 'retainedProps, 'action) => unit)
  | UpdateWithSideEffects(
      'state,
      self('state, 'retainedProps, 'action) => unit,
    )

~ reducer: (action, state) -> NoUpdate
```

action

```
type action =
  | DoIt;

let make = _children => {
  /* the other stuff is still here! */
  reducer: (action, _state) =>
    switch (action) {
    | DoIt => ReasonReact.NoUpdate
    },
};
```

types

```
reducer: ('action, 'state) => update('state, 'retainedProps, 'action)

update('state, 'retainedProps, 'action) =
  | NoUpdate
  | Update('state)
  | SideEffects(self('state, 'retainedProps, 'action) => unit)
  | UpdateWithSideEffects(
      'state,
      self('state, 'retainedProps, 'action) => unit,
    )
```

```
reducer: ('action, 'state) => update('state, 'retainedProps, 'action)

update('state, 'retainedProps, 'action) =
  | NoUpdate
  | Update('state)
  | SideEffects(self('state, 'retainedProps, 'action) => unit)
  | UpdateWithSideEffects(
      'state,
      self('state, 'retainedProps, 'action) => unit,
    )

~ reducer: (action, state) -> Update(state)
```

action

```
type state = {greeting: string};

type action =
  | DoIt;

let make = _children => {
  initialState: () => {greeting: "hi"},
  reducer(action, state) => switch (action) {
  | DoIt => ReasonReact.NoUpdate
  },
};
```

action

```
type state = {greeting: string};

type action =
  | DoIt;

let make = _children => {
  initialState: () => {greeting: "hi"},
  reducer(action, state) => switch (action) {
  | DoIt => ReasonReact.Update({...state, greeting: "yo"})
  },
};
```

types

```
reducer: ('action, 'state) => update('state, 'retainedProps, 'action)

update('state, 'retainedProps, 'action) =
  | NoUpdate
  | Update('state)
  | SideEffects(self('state, 'retainedProps, 'action) => unit)
  | UpdateWithSideEffects(
      'state,
      self('state, 'retainedProps, 'action) => unit,
    )

~ reducer: (action, state) -> SideEffects(self -> unit)
```

action

```
type state = {greeting: string};

type action =
  | DoIt;

let make = _children => {
  initialState: () => {greeting: "hi"},
  reducer: (action, _state) =>
    switch (action) {
    | DoIt => ReasonReact.SideEffects((self => Js.log(self.state.greeting)))
    },
};
```

types

```
reducer: ('action, 'state) => update('state, 'retainedProps, 'action)

update('state, 'retainedProps, 'action) =
  | NoUpdate
  | Update('state)
  | SideEffects(self('state, 'retainedProps, 'action) => unit)
  | UpdateWithSideEffects(
      'state,
      self('state, 'retainedProps, 'action) => unit,
    )
```

types

```
reducer: ('action, 'state) => update('state, 'retainedProps, 'action)

update('state, 'retainedProps, 'action) =
  | NoUpdate
  | Update('state)
  | SideEffects(self('state, 'retainedProps, 'action) => unit)
  | UpdateWithSideEffects(
      'state,
      self('state, 'retainedProps, 'action) => unit,
    )
```

types

```
reducer: ('action, 'state) => update('state, 'retainedProps, 'action)

update('state, 'retainedProps, 'action) =
  | NoUpdate
  | Update('state)
  | SideEffects(self('state, 'retainedProps, 'action) => unit)
  | UpdateWithSideEffects(
      'state,
      self('state, 'retainedProps, 'action) => unit,
    )
```

types

```
reducer: ('action, 'state) => update('state, 'retainedProps, 'action)

update('state, 'retainedProps, 'action) =
  | NoUpdate
  | Update('state)
  | SideEffects(self('state, 'retainedProps, 'action) => unit)
  | UpdateWithSideEffects(
      'state,
      self('state, 'retainedProps, 'action) => unit,
    )

~ reducer: (action, state) -> UpdateWithSideEffects(
    state,
    self -> unit
  )
```

action

```
type state = {greeting: string};

type action =
  | DoIt;

let make = _children => {
  initialState: () => {greeting: "hi"},
  reducer: (action, state) =>
    switch (action) {
    | DoIt =>
      ReasonReact.UpdateWithSideEffects(
        {...state, greeting: "yo"},
        (self => Js.log(self.state.greeting)),
      )
    },
};
```

okay, now what?

how do we use the reducer?

action

```
type state = {greeting: string};

type action =
  | DoIt;

let make = _children => {
  initialState: () => {greeting: "hi"},
  reducer: (action, state) =>
    switch (action) {
    | DoIt =>
      ReasonReact.UpdateWithSideEffects(
        {...state, greeting: "yo"},
        (self => Js.log(self.state.greeting)),
      )
    },
  render: self => <button onClick={_event => self.send(DoIt)} />,
};
```

mutation? 🤨

refs

```
let mutable = ref(5);

mutable := 4;
```

refs

```
let mutable = ref(5);

mutable := 4;
```

refs

```
let mutable = ref(5);

mutable := 4;

let immutable = ^mutable;
```

a time component, pt1

```
type state = {
  timerId: ref(option(Js.Global.intervalId)),
  time: int,
};

type action =
  | Tick;

let component = ReasonReact.reducerComponent("Component");
...
```

a time component, pt2

```
...
let make = _children => {
  ...component,
  initialState: () => {timerId: ref(None), time: 0},
  didMount: self =>
    self.state.timerId :=
      Some(Js.Global.setInterval(() => self.send(Tick), 1000)),
  willUnmount: self =>
    switch (self.state.timerId^) {
    | Some(id) => Js.Global.clearInterval(id)
    | None => ()
    },
  ...
```

a time component, pt3

```
...
reducer: (action, state) =>
  switch (action) {
    | Tick => ReasonReact.Update({...state, time: state.time + 1})
    },
  render: self => ReasonReact.string(string_of_int(self.state.time)),
};
```

a time component

```
type state = {
  timerId: ref(option(Js.Global.intervalId)),
  time: int,
};

type action =
  | Tick;

let component = ReasonReact.reducerComponent("Component3");

let make = _children => {
  ...component,
  initialState: () => {timerId: ref(None), time: 0},
  didMount: self =>
    self.state.timerId :=
      Some(Js.Global.setInterval(() => self.send(Tick), 1000)),
  willUnmount: self =>
    switch (self.state.timerId^) {
    | Some(id) => Js.Global.clearInterval(id)
    | None => ()
    },
  reducer: (action, state) =>
    switch (action) {
    | Tick => ReasonReact.Update({...state, time: state.time + 1})
    },
  render: self => ReasonReact.string(string_of_int(self.state.time)),
};
```

a time component

```
type state = {
  timerId: ref(option(Js.Global.intervalId)),
  time: int,
};

type action =
  | Tick;

let component = ReasonReact.reducerComponent("Component3");

let make = _children => {
  ...component,
  initialState: () => {timerId: ref(None), time: 0},
  didMount: self =>
    self.state.timerId :=
      Some(Js.Global.setInterval(() => self.send(Tick), 1000)),
  willUnmount: self =>
    switch (self.state.timerId^) {
    | Some(id) => Js.Global.clearInterval(id)
    | None => ()
    },
  reducer: (action, state) =>
    switch (action) {
    | Tick => ReasonReact.Update({...state, time: state.time + 1})
    },
  render: self => ReasonReact.string(string_of_int(self.state.time)),
};
```

a time component

```
type state = {
  timerId: ref(option(Js.Global.intervalId)),
  time: int,
};

type action =
  | Tick;

let component = ReasonReact.reducerComponent("Component3");

let make = _children => {
  ...component,
  initialState: () => {timerId: ref(None), time: 0},
  didMount: self =>
    self.state.timerId :=
      Some(Js.Global.setInterval(() => self.send(Tick), 1000)),
  willUnmount: self =>
    switch (self.state.timerId^) {
    | Some(id) => Js.Global.clearInterval(id)
    | None => ()
    },
  reducer: (action, state) =>
    switch (action) {
    | Tick => ReasonReact.Update({...state, time: state.time + 1})
    },
  render: self => ReasonReact.string(string_of_int(self.state.time)),
};
```

a time component

```
type state = {
  timerId: ref(option(Js.Global.intervalId)),
  time: int,
};

type action =
  | Tick;

let component = ReasonReact.reducerComponent("Component3");

let make = _children => {
  ...component,
  initialState: () => {timerId: ref(None), time: 0},
  didMount: self =>
    self.state.timerId :=
      Some(Js.Global.setInterval(() => self.send(Tick), 1000)),
  willUnmount: self =>
    switch (self.state.timerId^) {
    | Some(id) => Js.Global.clearInterval(id)
    | None => ()
    },
  reducer: (action, state) =>
    switch (action) {
    | Tick => ReasonReact.Update({...state, time: state.time + 1})
    },
  render: self => ReasonReact.string(string_of_int(self.state.time)),
};
```

a time component

```
type state = {
  timerId: ref(option(Js.Global.intervalId)),
  time: int,
};

type action =
  | Tick;

let component = ReasonReact.reducerComponent("Component3");

let make = _children => {
  ...component,
  initialState: () => {timerId: ref(None), time: 0},
  didMount: self =>
    self.state.timerId :=
      Some(Js.Global.setInterval(() => self.send(Tick), 1000)),
  willUnmount: self =>
    switch (self.state.timerId^) {
    | Some(id) => Js.Global.clearInterval(id)
    | None => ()
    },
  reducer: (action, state) =>
    switch (action) {
    | Tick => ReasonReact.Update({...state, time: state.time + 1})
    },
  render: self => ReasonReact.string(string_of_int(self.state.time)),
};
```

a time component

```
type state = {
  timerId: ref(option(Js.Global.intervalId)),
  time: int,
};

type action =
  | Tick;

let component = ReasonReact.reducerComponent("Component3");

let make = _children => {
  ...component,
  initialState: () => {timerId: ref(None), time: 0},
  didMount: self =>
    self.state.timerId :=
      Some(Js.Global.setInterval(() => self.send(Tick), 1000)),
  willUnmount: self =>
    switch (self.state.timerId^) {
    | Some(id) => Js.Global.clearInterval(id)
    | None => ()
    },
  reducer: (action, state) =>
    switch (action) {
    | Tick => ReasonReact.Update({...state, time: state.time + 1})
    },
  render: self => ReasonReact.string(string_of_int(self.state.time)),
};
```

an introduction to ReasonReact

pontus nagy

@p1xelHer0

thanks for listening ❤️